University Degree on Informatic Engineering

2017/2018

Bachelor Thesis

Active self-balancing camera system design and implementation using Arduino (Specific)

Adrian Pappalardo

Tutor: Javier Fernandez Muñoz

Abstract

This project describes the creation of an active stabilization system for cameras using Arduino systems as a controller. The system is intended to be used without any direct interaction from the user, instead, any interaction performed with the device will be automatically assessed and used to the system's working environment.

The system is not only conformed by the Arduino controller but also by all the electronic and robotic components that will make the physical object interact with the user. As well as including a set of components for the software used to program the controller in real time.

Index

Picture Index

Figure Index

Table Index

1. Introduction

General Vision

The project consists on developing a system capable of nullifying user movements while holding the device with the objective of stabilizing a camera positioned at the center of gravity of the device. This system is active, that means that there is a controller actively listening for movement changes and adjusting in real time a set of motors that will keep the camera balanced always.

This project involves a set of structural challenges, as well as hardware and software challenges. These set of challenges divide the system into three components that are related and dependent, but they are entire systems on their own, thus they have been treated in such a way.

Motivation

Due to the increase in technology in recent years, we have become able to do amazing things with the assistance of robotic machines when precision is needed. For some devices even with built-in robotic enhancements, due to the construction requirements for mass production and for end-user experiences, suffer from limitations on their capabilities, still leaving some space for improvement on third-party technologies that add to the original construction.

The concept of stabilizing objects is not new, it was originally achieved by using a passive gimbal which uses gravity and equilibrium to stabilize motion produced by its environment, yet this method can only support up to certain accelerations and certain movements, in addition, gravity only affects 2 axes of motion in a gimbal adding the need of an addition mechanism to control the third axis.

Given the technology available when the first gimbals arose it was not possible to control the movement of the third axis, but it was possible to mitigate the effects of the rotation by trying to nullify movement using bearings. The effect was the best you can achieve when using analog technology, but you still needed to precisely calibrate the system and it still had its limitations regarding the geometry of the objects and their center of mass, that is, because if the center of mass of the object was higher than the object itself, the system would not be able to support the weight as it goes in the same direction of the gravity that the system is trying to stop.

For this purpose, the utilization of robotics fixes these issues by granting active stabilization using sensors and motors to support higher weights and off-centered centers of gravity while also adding 3 axis manipulations. It also adds additional features to smooth the results and allow for a better result. This system solves these issues but at a cost, they need power and enough force to fight against the object that they are trying to stabilize, they also are considerable more expensive than the analog counterpart and are considerably more complex to achieve in a degree that is meaningful to use.

Goals

The man goal of this thesis is to design and develop an active stabilization system for a small camera using real-time programming in Arduino to try to get good results with a budget that is considerably lower than current technologies using the least amount of effective space as possible for its development.

The secondary goals include: to limit the complexity of the system to a degree that is executable by an Arduino controller in real-time, this implies getting knowledge about Arduino programming, and how the chipset works, its limitations and features and all the time-controlled functions that are required to synchronize everything. I will also be discussing knowledge regarding robotics for the sensor data extraction and the way motors can be controlled in these systems. Finally, I will be getting in depth on the real time, mathematical and physical formulas that were applied for managing and cleaning all the data for its use in the system.

We will be considering structural decisions for the system design that are crucial for the development of the project as physics has an incredible role in the system.

To fully define the objectives, I will present them in the following list:

- Make a 3-axis gimbal using only one Arduino Nano.
- Use Brushless motors to operate the system.
- Use only one gyroscope/accelerometer sensor without a magnetometer.
- Design a structure that supports a Sony QX10 compact camera.
- Design a completely energy independent system.
- Do not use any pre-built components aside from the Arduino itself, the sensor and the motors.

We will later discuss if these objectives were met and in which ways they were achieved

Development Phases

The project development structure is composed by 4 phases that will be described below.

Phase 1 – Viability Study

In this phase, the current technologies are analyzed in terms of usability and pricing, to see if the idea of the project is feasible. With this information, the different platforms available are analyzed and selected based on an initial set of basic requisites.

Phase 2 – System Requisites

This phase is for understanding how the system should work, thus developing a set of user requirements followed by a set of system requirements for the system to work in its intended environment.

Phase 3 – Design and Architecture

With all the platforms decided and the system requisites set, the design and architecture of the system is developed to ensure that all the design choices are correlated to the system requisites.

Phase 4 – Development and Testing

This system can be modeled in five different sub phases, all of which have been developed iteratively with significant changes on each iteration. Since these phases are independent from each other, each change done on one of the phases would not affect the functionality of the rest of the phases, even after finalizing each of them.

- Sub-Phase 1: Sensor Data Acquisition & Treatment
- Sub-Phase 2: Motor Movement
- Sub-Phase 3: Structure
- Sub-Phase 4: Real Time System

Since the development of this project follows an iterative approach, testing has been performed at the end of each sub-phase cycle. Feedback from this testing phase is used for the next iterations allowing also for changes in design and/or architecture when needed.

Document Structure

Introduction

In this section we explain the problem that the system is trying to solve, as well as the system itself, its components and the way it works. It also enumerates the different steps that went along the process of making the project.

This is the section in charge of stating the organization of this document and of the project.

State of the art

This section analyzes the market and the technologies that are already implemented for solving the issue.

It also enumerates different projects that has been done in the past with similar requirements from the open source community. It compares those projects with each other and with the system that we are trying to implement, stating pros and cons for using our system against theirs.

Analysis

This section describes the analysis process that the project went through for it to be completed.

In this part, the User Requirements and the System Requirements are clearly defined for posterior use in the project, alongside the use cases

Design

This process includes current technologies, ways of accessing these technologies, documentation available for these technologies and viability of the project itself.

The design part oversees selecting the correct technologies and creating a structure in which the development will be based.

Development

This section describes in detail all the development phases and what went into them. Each phase is iterative, that means that the feedback from the testing results is used to improve the next iteration of the system. This process is repeated until the specifications are met for each of the components of the system.

This section is divided into different components, each of which denote a sub phase of development. These sub phases are:

- Sensor data acquisition & treatment
- Motor movement
- Structure
- Real time system

Testing

This section oversees testing each component to verify that they fulfill the system functional requirements.

Planning and Cost Analysis

This section analyses the market for our product, including competitors, target population and pricing, as well as profitability and economic figures related to all the direct and indirect costs that went through the creation of the project. Finally, it describes the process of inserting the product into the market.

Conclusions and future work

This section contains the traceability matrix comparing the acquired results with the given requirements to see if it fulfills every one of them.

Additionally, the section describes the results of the project, personal achievements as a result of developing the project and future improvements for this development.

2. State of the Art

The term stabilization refers to counteracting a movement with an equivalent opposite force so that the result is a null movement.

Originally these systems were developed using gravity, since it was a very effective way of balancing movements using the force of nature. These contraptions are called gimbals and to this day they are still in use due to its energy independence property.

Nowadays nonetheless, the technology has evolved in such a way that it is more convenient to use robotics to assist a normal gimbal. This is where the active stabilization term comes into play. Active stabilization refers to a system that actively looks for movement and uses motors to emit an equivalent opposite movement that counters the original one. As a result, the system generates a stable surface with no perceivable movement. This was possible because of the introduction of gyroscopes, which are sensors capable of measuring angular velocity on the system that they are attached to. These new systems are very precise, fast and affordable. Their use is very broad, most notably they are used on vehicles to always have a context of the ground and speed,

lately it has been more notably used on portable devices such as smartphones and tablets, but their intent is to be used as a multifunctional tool for developers to use.

Active stabilization systems are usually developed as embedded systems and they are closed environments since most of them are programmed in a hardware level. Nowadays these end-user devices are very expensive due to the amount of development that the companies must undergo to make the system work properly, even though the components themselves are very cheap.

We will be focusing our attention on systems that are meant to stabilize cameras, although sometimes these systems double as vehicle stabilization, for example, some drones use the same sensor to stabilize the camera and the drone itself.

Camera types

To better understand how to stabilize a camera and the different technologies that are used for each one of them, it is better to analyze first what are the differences between current camera technologies mainly focusing on size, weight, and use cases.

Sports Cameras

Picture 1 - Sports camera – GoPro Hero 5 Session

These cameras are usually the smallest type of cameras, used in very harsh environments. Due to this type of usability, they tend to be very robust, they usually do not include a screen and they are very simple to use with most of them only containing a power button and a start-stop button.

Functionality is usually limited to recording videos since they usually do not have a viewfinder other than a smartphone in some cases.

Given all these conditions, these cameras are meant to be stabilized, but it usually depends on what mounting points does the camera have since the stabilization system would have to support the same kind of harsh environments as the camera.

Since they are meant to be mounted they are very compact and light, more information can be seen in the following table:

Size	40x40mm up to 80x80 mm
Weight	From 80gr to 120gr
Use case	Sport videography. Always mounted
Stabilization	Very necessary
Price	75€ to 500€

Table 1 - Sport cameras details

Note that these values are approximates and they do not represent the real range of all the cameras under this category.

Compact Cameras

Under this category there are three distinct sub-types,

- Smartphone assisted
- Compact cameras
- Mirrorless cameras

Each of these types has a different purpose, size and weight so each sub-type will develop its own comparison table.

Smartphone Assisted

Picture 2 - Smartphone assisted camera – Sony QX1

These cameras are a middle term between sports cameras and standard compact cameras, however, the use case is very different. These cameras are used when the user needs a fast way to take better pictures than the ones that their smartphones can provide and still have the result be saved in the memory of their smartphone. Therefore, size is usually only limited to the width of a medium sized smartphone and weight is not usually a concern, as well as durability, although portability is still an important factor.

The reason why these cameras need a smartphone to operate is because they do not feature a screen or camera controls of any type aside from the basic zoom and a button to take the picture or start a video.

There are different versions of these sub-type of cameras, ranging from all-in-one solutions to a more advanced 3 staged camera with each stage being the lens, the body and the mounting bracket for the smartphone.

In terms of weight and size, the following table summarizes the approximate ranges:

Size	60x60mm up to 80x80 mm
Weight	From 100gr to 220gr
Use case	Smartphone photography
Stabilization	Needed for video recording or taking night photos
Price	200€ to 350€ (w/o lens)

Table 2 - Smartphone assisted cameras details

Compact cameras

Picture 3 - Compact camera – Sony DSC-RX100

These cameras are intended to be easy and fast to use, while being portable and light. They usually have a limited amount of functions that the user can modify, such as zoom and exposure, the rest are intended for automated use in most cases.

These type of cameras uses a screen to command the camera and see the composition of the shot before taking the picture. They also feature non-removable lenses since the intention is to have a very compact and portable system.

The use case for this type of cameras is most of the consumer level users and some photography enthusiasts at the entry level, since they are the easiest type to use and they usually give good quality pictures when compared to more advanced cameras.

The different weights and sizes can be seen in the following table:

Size	90x50mm to 100x60mm
Weight	100gr to 240gr
Use case	Everyday photography
Stabilization	Not very needed since the uses for these cameras are very simple
Price	50€ to 350€

Table 3 - Compact cameras details

Mirrorless Cameras

Picture 4 - Mirrorless camera – Sony A7 III

These are the most advanced type of compact cameras. They feature 2/3 frame or full-frame sensors, making them have the exact same sensor types as a DSLR but with a compact size. This is achieved by removing the prism at the top of a DSLR and replacing it with a digital viewfinder.

These cameras achieve excellent quality and all the features for professional photography are user editable and they feature interchangeable lenses for different use cases.

The use case for these cameras is enthusiast or entry level professional videography due to their size and video quality. These cameras are also used for professional photography.

Since these cameras are more targeted to advanced photography, size and weight are not that much of a concern, as can be seen in the following table:

Size	110x60mm to 130x100mm
Weight	270gr to 670gr
Use case	Professional photography and videography
Stabilization	Very used due to the small size and video quality
Price	350€ to 2.500€ (w/o lens)

Table 4 - Mirrorless cameras details

DSLR

Picture 5 - DSLR Camera – Canon 100D

These are the professional level cameras; these cameras feature a mirrored viewfinder and full-frame sensors. They are full featured and have manual controls for every setting. They also feature interchangeable lenses.

Size and weight are not the focus of these cameras, so they tend to be big and heavy.

The use case for this type of devices is professional photography and videography, although they are mostly used for the first use case.

Size	130x100mm to 160x170mm
Weight	475gr to 1530gr
Use case	Professional photography and videography
Stabilization	Very important for professional videography
Price	550€ to 5.500€ (w/o lens)

Table 5 - DSLR cameras details

Medium Size Cameras

Picture 6 - Medium size camera – Hasselblad H6D

These cameras are the most advanced on the camera spectrum, they are used for very high-quality high-end photography such as marketing for big companies and large-sized prints.

The sensor in these devices is interchangeable and are usually twice as big as a full-frame DSLR camera, allowing for considerable more light input into the sensor.

They are very modular since the sensor can be changed as well as the lens, the handle and the viewfinder, several add-ons are available for these cameras.

They are used with stabilization systems whenever a video is captured due to the expected result being of the highest quality possible.

They are very heavy and big since the focus is picture quality and not portability.

Size	280x280mm
Weight	2000gr
Use case	High-end professional photography and videography
Stabilization	Very important for professional videography
Price	10.000€ to 35.000€ (w/o lens)

Table 6 - Medium size cameras details

Current Technologies

The current state of the art for these devices is still divided into 2 main categories; Passive and Active systems. We will discuss some examples of this technologies and their differences below.

For comparing these technologies, we will use a table on each of the technologies comparing the following aspects:

- **Controller**: Describes the type of controller utilized in case that it uses a standard consumer-level controller or if it uses a proprietary one.
- **Motor type**: Denotes what type of motor the system uses.
- **Mounting**: Describes the means necessary to operate the device, if it uses one or two hands or more mounting points.
- **Camera support**: The size and weight of the cameras that can support as a maximum. Usual camera types to be compared will be: Sports, Compact, Mirrorless, DSLR, Medium size
- **Preparation**: Investigates the steps necessary to prepare the device for operation.
- **Cost**: The range of prices of these devices.

Passive Stabilization Systems

Picture 7 – Passive stabilization system - FOTOWELT HD-2000

This technology uses the power of gravity to nullify movement. The way this system works is by having a handle that must be placed manually in the center of gravity of the system. The handle itself has separate axis for each rotation, this means that any intent of rotation will instead be translated into positional movement, since there is not enough friction between the moving components on the handle and the rest of the device. Given that rotation often comes in small quantities it becomes small enough for the stabilization system on the camera itself to correct any further movement.

It is important to note that cameras only come with positional stabilization systems due to limitations on size for rotational stabilization.

This technology is very affordable since it is purely mechanical and with a very simple design. They are nonetheless a lot more complicated to start working with, since every time the user must place the camera, the system must be calibrated until everything is perfectly balanced, otherwise the system will not work.

To further analyze and compare this system, a table will be used:

Controller	The system does not use a controller
Motor Type	The system does not use motors
Mounting	One handed use, using a handle
Camera Support	The system can operate with any size of cameras at any weight depending on the counterweight used. Note that the more weight added to the system the better the system will perform in terms of movement but the worse in terms of speed, also consider the holding method since a heavier camera might be needed to be operated with two hands
Preparation	This system requires a lot of preparation from the user, given how it works, a precise balance must be acquired before using. For this, the counterweight must be of at least same effective weight as the camera but the heavier the better and the center of gravity of the complete system must be exactly at the handle
Cost	The cost for these technologies varies from 50€ for the most affordable versions, to around 120€ for a full featured version

Table 7 - Passive stabilization system details

Picture 8 - Passive stabilization system - Opteka X-Grip

There are other kinds of passive stabilizations called static stabilization systems such as the one seen in the picture above. In this case the stabilization is done by ergonomics, to be more precise, it is done by changing the position in which the user grabs the device to the most optimal in terms of stabilization, it also forces the user to grab the device in a specific way that has been studied to reduce unwanted movement.

To further analyze and compare this system, a table will be used:

Controller	The system does not use a controller
Motor Type	The system does not use motors
Mounting	One handed use, using a handle
Camera Support	The system can operate with any size of cameras at any weight but note that the size is usually not expandable, so the internal size of the system must be considered in comparison with the camera to be used. Usual ranges of cameras are from Mirrorless to DSLR
Preparation	This system does not require any preparation
Cost	The cost for these technologies varies from 50€ for the most affordable versions, to around 120€ for a full featured version

Table 8 - Static stabilization system details

Active Stabilization Systems

In this category we can find several types of systems all depending on the size of the camera to be used and on the mounting system used to attach or grab the device.

Small size

Picture 9 – Active stabilization system - DJI Inspire 2

They are meant to be used with sport cameras. These cameras are often used for embedded applications into other equipment such as drones or helmets, they are very compact because they lack power, but they have enough to satisfy the needs of a small camera.

We can find several options to mount our own small camera into existing drones as well as we can find drones with proprietary versions to be even more compact and light.

The way this systems work is either by having their own separate gyroscope sensor, or by using the one that is already being used to stabilize the host device, for example in the case of a drone it would be more efficient to use the same sensor to stabilize the drone and the camera at the same time since adding more sensors would result in all of them sending almost identical information as the main one.

In addition to the sensor system they are composed by a controller and a set of motors, usually one per axis. The technology used in these motors is always brushless motors since they fulfill the task in the most reliable and smooth way possible.

These options are very cost effective since the electronics used to create these devices is very inexpensive and easy to acquire and since they are very compact, they are not very complex in their structure.

To further analyze and compare this system, a table will be used:

Controller	• Motor controller using the host sensor, for example in the case of a drone, using the drone's own gyroscope • Standard gimbal controller
Motor Type	Low speed, medium torque brushless motors
Mounting	Mounted on a host device
Camera Support	These systems support small cameras such as sports cameras
Preparation	This system does not require any preparation
Cost	The cost for these technologies varies from 75€ for the most affordable versions, to around 100€ for a sturdier version. Note that this cost does not include the cost of the host device

Table 9 - Active stabilization system – small size details

Medium size

Picture 10 - IKAN MS1

These systems are meant usually to be handled by a person directly. They are an enclosed unit containing all the components into one portable package that can be operated with only the need of one hand. They are powerful enough to handle compact

cameras such as point and shoot cameras or mirrorless cameras, in the latter case, the system needs to be adjusted so the center of gravity of the camera, including the lens, is equal to the center of gravity of the socket that holds the camera. This is an easy process to do and it does not need to be perfect, it aims to assist the system by releasing some stress generated by the force of gravity on an unbalanced camera.

The way this systems work is by having a set of motors that are rated for a maximum camera weight and a set of sensors that can be placed on the camera socket or on the handle. The sensor would then tell the angle of the camera in comparison with the ground and tell the motors to counter that movement. This works in several positions usually, it is not mandatory to hold the device vertical for the system to operate, some systems are meant to be used horizontally for added passive stability, like the one in Picture 2.

This option has affordable options with low power for smaller cameras, and more powerful versions for heavier cameras at a higher cost.

To further analyze and compare this system, a table will be used:

Controller	Proprietary controller
Motor Type	Medium speed, medium torque brushless motors
Mounting	They use a one-handed handle that can be used both vertically and horizontally
Camera Support	These systems support compact cameras, from smartphone assisted to advanced compact cameras
Preparation	This system requires a small balancing of the camera
Cost	The cost for these technologies varies from 100€ for the most affordable versions, to around 350€ for a sturdier, full featured version.

Table 10 - Active stabilization system - medium size details

Large size

Picture 11 - DJI Ronin-MX

These systems are meant to be held by a user dedicated to holding the camera due to its weight and size, it usually requires both hands and a mount to the chest in extreme cases. Given the size of the system it can hold the biggest cameras available on the market, they also imply a harder structure, more power and speed, and an increased battery life when compare to the other variants of active stabilization. Some also accept the addition of several upgrades to control the camera remotely such as programmable focus or buttons to control the camera from the handles.

The way this systems work is by using several sensors to determine the user movement and the camera position and using a set of 3 or more motors to counteract that movement, these motors are brushless and high torque. In this case camera positioning and weight balance is recommended but not mandatory, since the motors can withstand most of the load. The structure is usually made from carbon fiber or aluminum.

These systems are the most expensive solution that can be purchased on the market.

To further analyze and compare this system, a table will be used:

Controller	Proprietary controller
Motor Type	High speed, high torque brushless motors, usually more than 3 to add more torque to the most stressed axes
Mounting	They have a structure that allows for one handed use with a horizontal handle and two-handed use with vertical handles. They also come with different mounting options such as a chest mount or even vehicle mounts for professional videography
Camera Support	These systems support medium to big sized cameras
Preparation	This system requires a small balancing of the camera
Cost	The cost for these technologies varies from 1000€ for the most affordable versions, to around 10.000€ for a sturdier, full featured version.

Table 11 - Active stabilization system - Large size details

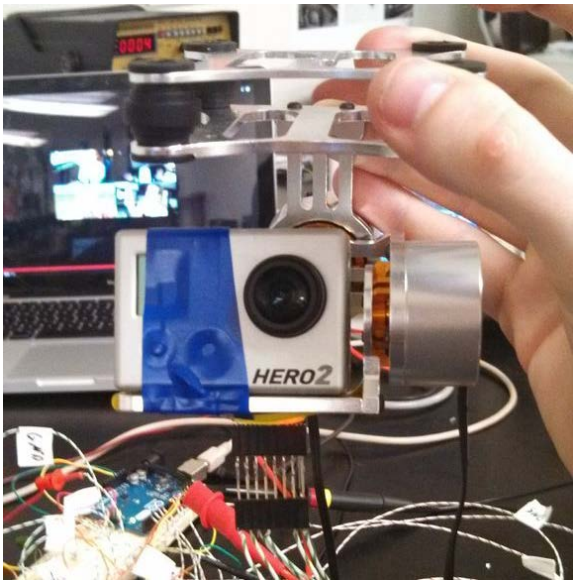
Micro-controller System Developments

Even though most of these systems are closed environments, some projects were made using Arduino, so anyone could contribute to the development and improvement of these systems, thus reducing the overall cost of creating a new system.

Some of the projects ended in success but there were some that failed, but they left a considerable amount of information for future works to improve.

We will describe next the most impactful Arduino developments that are related to the project.

2-axis brushless stabilization system for GoPro



Picture 12 - Brushless Gimbal with Arduino - by Jonan Grubb

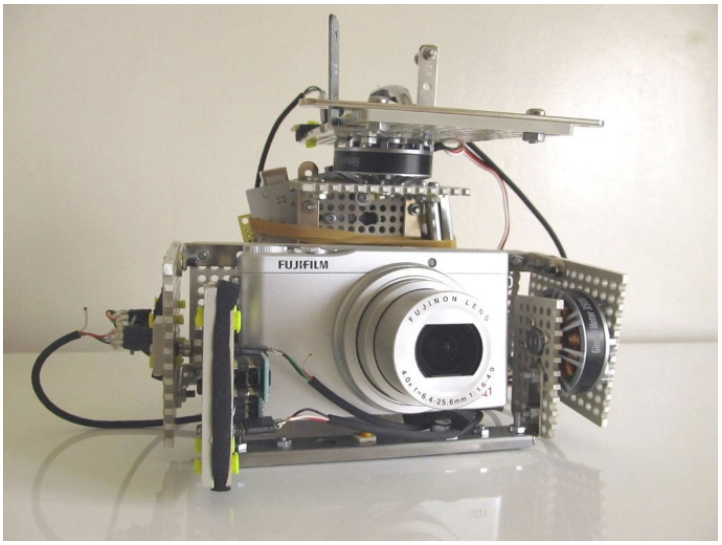
This project was published on instructables [3]. It is an active stabilization gimbal for small sized cameras that uses an Arduino Uno and a MPU6050 gyroscope along with brushless motors without independent controllers.

The project intended to use the Arduino itself as a controller for the brushless motors, which succeeded on doing, but it also used premade code and hardware for the rest of the components. In the end, the different components did not communicate properly, and the system did not work as intended, so it was discarded.

Controller	Arduino
Motor Type	High speed, low torque brushless motors
Mounting	This system used a standard drone mount, but it was not intended to be mounted
Camera Support	This system support sport cameras, specifically GoPro Hero 2
Preparation	This system does not require preparation
Axis of rotation	2 Axis
Power supply	A USB cable to a computer

Table 12 - Brushless Gimbal with Arduino details

3-Axis brushless stabilization system without external controllers



Picture 13 - The Making of a DIY Brushless Gimbal with Arduino - by ArduinoDeXXX

This development is based on the previously described project with the difference that this one did succeed in creating a working version that is useful in the real world.

This is also a development published in Instructables [2] and it had the following objectives:

- It had to support a camera bigger than a GoPro.
- It must be as simple as possible without the use of any external pre-built component aside from the Arduino, the Sensors and the motors.
- Built from scratch with no use of external coding.

The project was originally intended to be a 2-axis servo gimbal, but it has proven to be insufficient given the requirements for this project, instead it ended as a 3-axis brushless motor gimbal with 4 sensors and an Arduino Mega.

The main challenges for this project were:

- Controlling a brushless using only an Arduino.
- Resolving the SPI interface issues between the sensors and the Arduino.
- Structural integrity affecting sensor data.
- Compensating movement precisely using compensation algorithms.

The design of this project considered several factors and attempts to try to achieve an acceptable result:

1. Firstly, the selection of the motor itself considered Servo, Brushed DC and Brushless DC motors. Note that we are comparing the cheap versions of each of these motors as bigger iterations often come with an increase on the quality and functionality.
    a. Servo motors proved to be too slow and weak for this purpose.
    b. Brushed DC motors were the first approach, failing due to its lack of precision and constant vibrations when trying to compensate movement, affecting in a big way the performance of the sensors.
    c. Finally, Brushless DC motors were chosen being strong and silent enough for the project with the only downside being the lack of a controller that could move the motor based on angles instead of RPM and, given the requirements, such controller cannot be used as it would be an external pre-built source, therefore the Arduino should be the controller for the motors, program that should be later developed on his own.

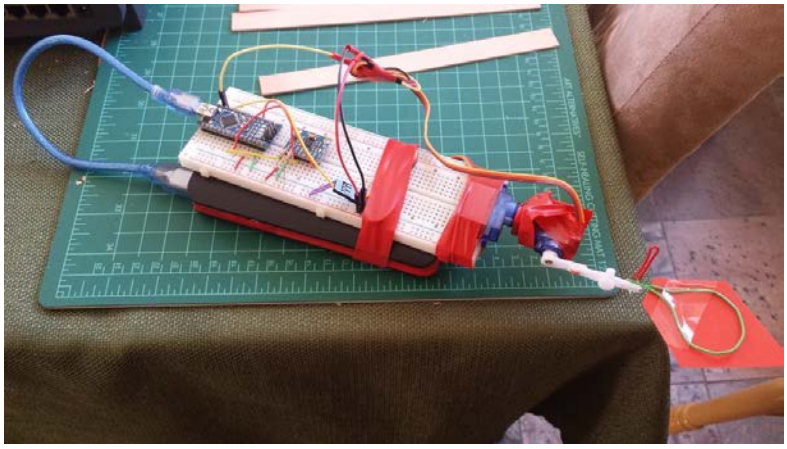
2. As a second instance, the sensors used, their position and amount should be determined.
    a. Sensor.
        i. There was a possibility on using the MPU6050 due to its noise-cancelling features, but as the requirement was not to use any external code, this addition was pointless, so it was not beneficial nor a suitable option for the project.
        ii. The sensor used is L3GD20 with its carriers.
    b. Position and Amount.
        i. Attaching it to the Structure, before any movement occurs is useful due to the isolation of noise coming from the movement of the motors, but it does not calculate a precise motion but an approximation on how the motor moved. This can be useful only if the motor is reliable enough.
        ii. On the other hand, attaching it into the camera gives a more precise calculation but it is vibration dependent because of the movement of the motors.
        iii. Finally, the chosen schema was to use one on the Structure and one on each of the axis giving a total of 4 sensors.

A summary of the project can be found in the following table

Controller	Arduino
Motor Type	High speed, high torque brushless motors
Mounting	The system does not have a mounting solution
Camera Support	This system supports compact cameras
Preparation	This system does not require preparation
Axis of rotation	3 Axis
Power supply	External battery or USB connection

Table 13 - The Making of a DIY Brushless Gimbal with Arduino details

2-Axis Servo stabilization system



Picture 14 - Gyro Stabilizer W/ Arduino and Servo - by woojay

This development has been published in instructables [1] and it is a simple stabilization algorithm that uses an Arduino nano, an MPU 6050 gyroscope and two servo motors attached one into another.

The design only allows for 2-axis stabilization and it does not support any weight, it's a mere representation on the capabilities of the device. The servos move slow but precisely and it generates a considerable noise. It is also energy independent as it is using an external battery to power the entire system.

This system works as a proof of concept as it cannot be used for anything but the principles for acquiring the data should remain the same if the motors change and the structure is upgraded.

Controller	Arduino
Motor Type	Servo motors
Mounting	The system does not have a mounting solution
Camera Support	This system does not support cameras
Preparation	This system does not require preparation
Axis of rotation	2 Axis
Power supply	External battery or USB connection

Table 14 - Gyro Stabilizer W/ Arduino and Servo details

Microcontrollers

“A microcontroller is a small, low-cost and self-contained computer-on-a-chip that can be used as an embedded system […] Microcontrollers usually must have low-power requirements since many devices they control are battery-operated. Microcontrollers are used in many consumer electronics, car engines, computer peripherals and test or measurement equipment. And these are well suited for long lasting battery applications. The dominant part of microcontrollers being used nowadays are implanted in other apparatus.” [8]

Classification

Microcontrollers can be classified based on

- Number of bits
  - 8 Bit
  - 16 Bit
  - 32 Bit
- Type of memory architecture
  - Harvard Memory Architecture Microcontroller
  - Princeton Memory Architecture Microcontroller
- Instruction set
  - Complex Instruction Set Computer (CISC)
  - Reduced Instruction Set Computer (RISC)

There are 4 types of microcontrollers

Microcontroller 8051

It is an 8-bit microcontroller with 40 pins and voltage control. It uses the C language with pre-compilation on the source of the code transmission. It is known to be very hard to program.

It uses a CISC architecture.

Renesas Microcontroller

Renesas is a low powered, high performance microcontroller which offers security and embedded safety mechanisms, making it the best option for automotive computation.

It is a 32-bit microcontroller using an enhanced Harvard CISC architecture to achieve its renown performance. It contains a total of 20 pins.

AVR Microcontrollers

The name stands for Alf-Egil Bogen and Vegard Wollan's RISC processor. It uses a modified Harvard RISC architecture with separate memories for data and program.

There is a very well know microcontroller in this category called the ATmega328, better known as Arduino, which is a 28-pin AVR microcontroller. It is very broadly used thus there is a considerable amount of information regarding how to code using this microcontroller and, although it uses the C language, it is very simple to use.

Arduino microcontrollers have an additional benefit which is that it is widely used for prototyping, thus its versions are either full boards with all the pin connections ready for prototyping or the single chip that can be used to create a custom board that fits the needs of the system.

It differs from 8051 microcontrollers in the sense that AVR controllers are faster, easier to program and it uses a RISC architecture instead of a CISC in the 8051 controllers, they also consume less power than the 8051 controllers.

PIC Microcontrollers

This is a high-performance RISC CPU using a Harvard architecture that can operate with a wide range of voltages. They are very flexible and have embedded features such as a sleep function, a Digital to Analog converter (DAC) and an Analog to Digital Converter (ADC), an Analog Comparator Module, Synchronous serial port with master mode and I^2C Master/Slave for multiple connections.

These devices are mainly used in smartphones, computer peripherals and audio accessories.

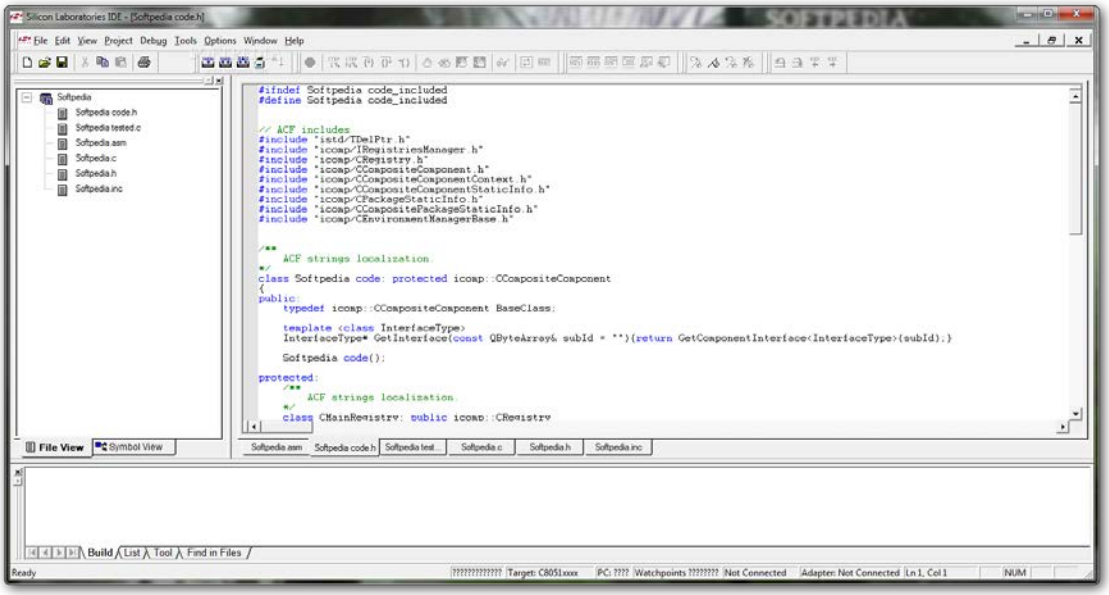
Software

The languages that are used to code in any microcontroller are the compiled ones, being the most famous the C and C++ languages.

Although there are several ways to program in the C/C++ languages, there are specific sets of software or environments that are ready to communicate with the microcontroller that is being used.

We will be viewing the 3 most used tools for flashing code into a microcontroller which are mainly assigned to a specific family of microcontrollers.

Silicon Labs IDE

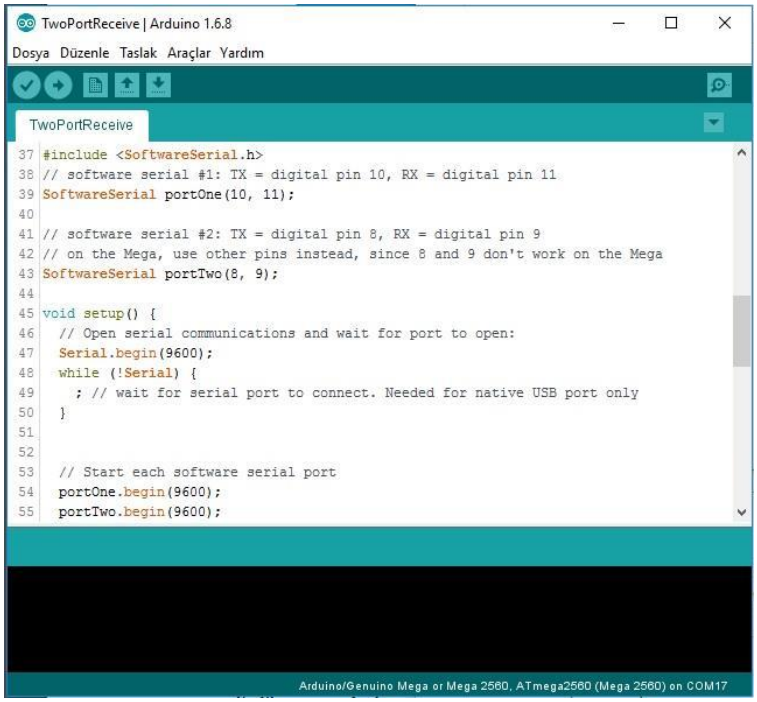


Picture 15 - Software - Silicon Labs IDE

The Silicon Labs Integrated Development Environment is a complete, stand-alone software program that includes a project manager, source editor, source-level debugger and other utilities. The IDE interfaces to third party development tool chains to provide system designers a complete embedded software development environment. The IDE supports the entire 8-bit microcontroller (MCU) portfolio.

This is a controller designed for all microcontrollers that feature an 8-bit architecture, thus it works with 8051 and AVR microcontrollers

Arduino IDE



Picture 16 - Software - Arduino IDE

This is an open source software to interface with ATmega microcontrollers, it adds a set of libraries that makes programming easy and high level, but it does not remove the low-level capacities of programming with C/C++ languages.

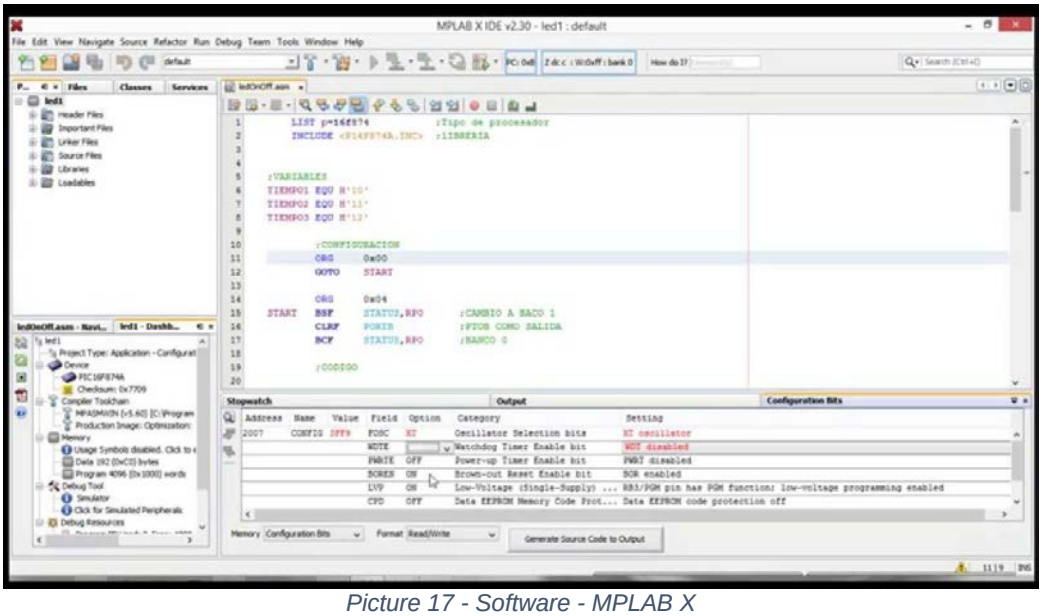
It supports contextual files as part of a project and it offers debugging for the code with highlights on the part of the code that emits the error.

It compiles the code and flashes it into the microcontroller using a COM port in the host operating system.

It comes with pre-built scenes for common situations and with online tutorials for learning how to use the software's tools. In addition, it has an online editor to test the code before flashing it into the microcontroller, reducing the risks of something going wrong.

It is compatible with Windows, MacOS and Linux and it is open source and editable to the needs of the project.

MPLAB X



Picture 17 - Software - MPLAB X

It is an IDE based on NetBeans from Oracle. It runs in Windows, MacOS and Linux and it is used to program PIC microcontrollers.

It also includes a compiler and a cloud version to test the code before flashing into the microcontroller.

It includes a powerful set of plugins and libraries to ease the programming of the code done in C or C++ and it manages all the code in self-contained projects.

It supports contextual files as part of a project and it offers debugging for the code with highlights on the part of the code that emits the error.

It compiles the code and flashes it into the microcontroller using a COM port in the host operating system.

Technologies summary

In this section all the technologies concerning cameras and stabilizers will be compared to make a more comparative approach to them, so they become easier to understand.

They are also known to have a very efficient way of coding, thus using less code in memory to execute the same tasks as other microcontrollers

To better understand the relation between stabilizers and cameras, a traceability matrix shows which stabilizers are meant for which type of cameras.

	Passive	Static	Active Small	Active Medium	Active Large
Sports			X		
Smartphone Assisted			X	X	
Compact	X			X	
Mirrorless	X	X		X	X
DSLR	X	X			X
Medium format					X

Table 15 - Cameras vs Stabilizers

To better understand the relation between microcontrollers and between microcontrollers and their software's, the following table compares all the options.

Microcontroller	Clock	Target Market	Complexity	Software
8051	8-bit	Deprecated	Complex	Silicon Labs IDE
Renesas	32-bit	• Industrial automation • Communication applications • Motor control applications • Test and measurement • Medical applications	Normal	Renesas Flash Utility
AVR	8-bit	• Home automation • Touch screen • Automobiles • Medical devices • Defense	Easy with libraries	Silicon Labs IDE, Arduino IDE
PIC	8-bit + 16-bit	• Smart phones • Audio accessories • Computer peripherals • Advanced medical devices	Easy with libraries	MPLAB X

Table 16 - Microcontrollers comparison

3. Analysis

In this section we will be describing the thought process and the decision tree used to choose the final components of the system, this does not intent to be a precise detailed description on the process of creating the system but of the design phases that happened along the way. For a more detailed description of each decision, refer to the Development section.

User Requirements

This section contains the description of the user requirements, that will be divided in two categories: Capability and Restriction.

The tables that will contain the description of the data will contain the following fields:

- **Name**: Identifiable name of the requisite, it must be a concise version of the description of the content.
- **ID**: ID of the user requirement. It must be unique for each requirement. The format for this field is UR_RR_XX where RR can take the values of CR for capability requirements or RR for restriction requirements, and XX is an incremental padded number identifying the requirement within its category.
- **Description**: Contains a detailed description on the requisite. It aims to be as clear as possible to avoid ambiguity and confusion.
- **Priority**: Indicates the priority of implementation of the requisite, this is meant to ease the task of organization during the implementation of the project.
- **Necessity**: Indicates how necessary the requirement is for the project, this is meant to add flexibility under time constraints if any.
- **Stability**: Indicates if the requirement can be subject to changes along the implementation of the project, or if it is a stable requirement that will not change.
- **Verifiability**: Indicates how verifiable is the introduction of the requirement into the system, that is, to check if the requirement is fulfilled.

Capability requirements

Capability requisites indicates which functions can or have to do the system for it to comply with the requisites.

	Active Stabilization		UR_CR_01
Priority	High	Necessity	High
Stability	Stable	Verifiability	High
Description	The system must behave as an active stabilization system with the use of electronics to assist stabilization.		

Table 17 - User Requirement UR_CR_01

Active Stabilization Axis of rotation			UR_CR_02
Priority	High	Necessity	High
Stability	Stable	Verifiability	High
Description	The system must perform the stabilization for all axis of rotation. Spatial movement should not be considered for stabilization purposes.		

Table 18 - User Requirement UR_CR_02

Z Axis			UR_CR_03
Priority	Medium	Necessity	Medium
Stability	Unstable	Verifiability	High
Description	The system must also have the ability to move the Z Axis manually.		

Table 19 - User Requirement UR_CR_03

Camera compatibility			UR_CR_04
Priority	High	Necessity	High
Stability	Stable	Verifiability	High
Description	The system must be able to function with a Sony QX100 camera.		

Table 20 - User Requirement UR_CR_04

Passive stabilization			UR_CR_05
Priority	Very low	Necessity	Very low
Stability	Unstable	Verifiability	Medium
Description	The system must operate under a passive system alongside the active one.		

Table 21 - User Requirement UR_CR_05

Feedback			UR_CR_06
Priority	High	Necessity	High
Stability	Stable	Verifiability	High
Description	The system must communicate its status to the operator via the use of LEDs:		

Table 22 - User Requirement UR_CR_06

Restriction requirements

Restriction requirements denote the limits of the system and sets conditions that must be met to reduce the uncertainty of the development.

Enclosed System			UR_RR_01
Priority	Medium	Necessity	High
Stability	Stable	Verifiability	High
Description	The system must be able to operate only within its own components, including power and communication, and the entire system must be enclosed in a single structural unit.		

Table 23 - User Requirement UR_RR_01

Portability			UR_RR_02
Priority	Medium	Necessity	High
Stability	Stable	Verifiability	High
Description	The system structure must be able to endure transportation and utilization in environments with different temperature and humidity conditions.		

Table 24 - User Requirement UR_RR_02

One hand use			UR_RR_03
Priority	Low	Necessity	Medium
Stability	Stable	Verifiability	Medium
Description	The system must be able to be operated with only one hand holding the device.		

Table 25 - User Requirement UR_RR_03

External components			UR_RR_04
Priority	Medium	Necessity	Low
Stability	Unstable	Verifiability	Medium
Description	The system must not use any external code or pre-made hardware controllers. Although it can use parts of external code, the result of the code must be proprietary, and the sources must be cited appropriately.		

Table 26 - User Requirement UR_RR_04

Speed			UR_RR_05
Priority	Medium	Necessity	Medium
Stability	Stable	Verifiability	Low
Description	The system must be responsive enough to counteract the user movement in real time and that a 30fps camera setting cannot see the corrections.		

Table 27 - User Requirement UR_RR_05

Noise			UR_RR_06
Priority	Low	Necessity	Low
Stability	Stable	Verifiability	Medium
Description	The system must perform without disturbing in more than 10db the audio of the camera microphone while recording.		

Table 28 - User Requirement UR_RR_06

Use Cases

This section will explain the different interactions between the stakeholders of the system. Since it is a system with a very low interaction, the use cases are limited to the different steps of the process of using the device.

The use cases will be modeled as tables containing the following fields:

- **Name**: Name identifying the use case, it must be unique, and it will be descriptive on what the use case resolves.
- **ID**: Unique identifier for the use case, it follows the format UC_XX where XX is an incremental padded number.
- **Actor**: External agent that interacts with the system.
- **Objective**: Action to be analyzed by the interaction of the actor with the system.
- **Pre-condition**: Conditions that must be met before the use case can be executed.
- **Scene**: Detailed description of the step that the actor must follow to complete the use case.
- **Alternative Scene**: Detailed description of the different paths that the use case can take, if any, and how to resolve them.
- **Post-condition**: Description of the result given that the use case is correctly completed.

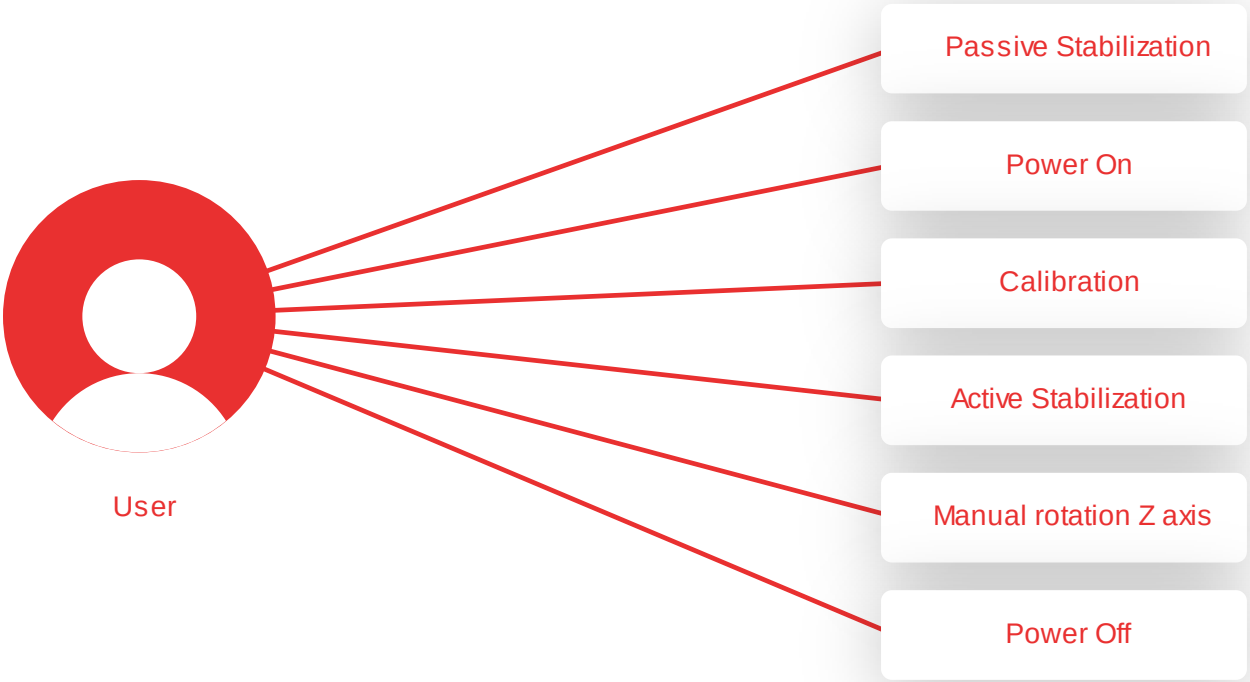


Figure 1 - Use Cases

Passive Stabilization	UC_01
Actor	User of the system
Objective	Stabilize the camera without the need of turning the device on
Pre-condition	• The system must be off • The camera must be in place and balanced
Scene	• Grab the device by the handle • Rotate the device on the X or Y axis of rotation
Alt. Scene	
Post-condition	The system will handle basic slow movements without much inertia, thus it will stabilize the camera

Table 29 - Use case UC_01 Passive Stabilization

Power On	UC_02
Actor	User of the system
Objective	Power the active system
Pre-condition	• The system must be off
Scene	• Place the device on a horizontal steady surface • Press the power button located in the battery attached to the back of the handle
Alt. Scene	
Post-condition	The system will have power and it will start the initialization processes

Table 30 - Use case UC_02 Power On

Calibration	UC_03
Actor	User of the system
Objective	Power the active system to the state of full functionality
Pre-condition	• The system must be on • The camera must be in place and balanced • The system must be placed in a steady horizontal surface
Scene	• Wait for the red lights located at the front of the handle to start to blink
Alt. Scene	
Post-condition	The system will be calibrated for full functionality

Table 31 - Use case UC_03 Calibration

Active Stabilization	UC_04
Actor	User of the system
Objective	Use the system to stabilize the camera on all axes
Pre-condition	• The system must be on • The system must be calibrated • The camera must be in place and balanced
Scene	• Grab the system by the handle • Rotate the device on any axis
Alt. Scene	
Post-condition	The system will counteract any movement on any of the axes on real time

Table 32 - Use case UC_04 Active Stabilization

Manual Rotation Z Axis	UC_05
Actor	User of the system
Objective	Rotate the Z axis manually to use the ‘Pan’ functionality
Pre-condition	• The system must be on • The system must be calibrated • The camera must be in place and balanced
Scene	• Grab the system by the handle • Use the joystick placed at the front of the handle to rotate the Z axis to the left or right
Alt. Scene	• Press on the joystick to set the coordinates back to the original position
Post-condition	The system will move the Z axis to point where the joystick has told it to do

Table 33 - Use case UC_05 Manual rotation Z axis

Power Off		UC_06
Actor	User of the system	
Objective	Power off the system	
Pre-condition	• The system must be on	
Scene	• Press the power button located in the battery attached to the back of the handle	
Alt. Scene		
Post-condition	The system will move the Z axis to point where the joystick has told it to do	

Table 34 - Use case UC_06 Power Off

System Requirements

The system requirements are meant to describe the system functionality. They are divided in Functional requirements and Non-Functional requirements.

They will be described in the form of tables containing the following fields:

- **Name**: Identifiable name of the requisite, it must be a concise version of the description of the content.
- **ID**: ID of the user requirement. It must be unique for each requirement. The format for this field is SR_RR_XX where SR represents that it is a system requirement, RR denotes if the requirement is a functional or a nonfunctional requirement, it can denote FR or NFR respectively and XX is an incremental padded number identifying the requirement within its category.
- **Description**: Contains a detailed description on the requisite. It aims to be as clear as possible to avoid ambiguity and confusion.
- **Priority**: Indicates the priority of implementation of the requisite, this is meant to ease the task of organization during the implementation of the project.
- **Necessity**: Indicates how necessary the requirement is for the project, this is meant to add flexibility under time constraints if any.

- **Stability**: Indicates if the requirement can be subject to changes along the implementation of the project, or if it is a stable requirement that will not change.
- **Verifiability**: Indicates how verifiable is the introduction of the requirement into the system, that is, to check if the requirement is fulfilled.
- **Dependencies**: Indicates what user requirement it is evaluating and trying to fulfill. Each User Requirement must be fulfilled by one or more System Requirement.

Functional Requirements

	Movement response		SR_FR_01
Priority	High	Necessity	High
Stability	Stable	Verifiability	Medium
Description	All axis must move automatically with the input of the sensors alone with the purpose of counteracting any movement detected by said sensor		
Dependencies	UR_CR_01, UR_RR_05		

Table 35 - System Requirement SR_FR_01

	Z Index movement		SR_FR_02
Priority	Medium	Necessity	Low
Stability	Unstable	Verifiability	High
Description	Z axis can also be moved manually, with the use of an input into the controller		
Dependencies	UR_CR_03		

Table 36 - System Requirement SR_FR_02

	Passive stabilization		SR_FR_03
Priority	Low	Necessity	Low
Stability	Unstable	Verifiability	Medium
Description	The structure of the system must be composed by a three degrees of freedom motorized enclosure that maintain a balance perpendicular with earth's vector without the need of external power. This case will only apply if the camera is mounted and balanced with respect to the ground		
Dependencies	UR_CR_05		

Table 37 - System Requirement SR_FR_03

	System feedback		SR_FR_04
Priority	Medium	Necessity	High
Stability	Stable	Verifiability	High
Description	The system must turn the LEDs • off if the system is off • static red when the system is on • blinking red when the system is calibrated and operational		
Dependencies	UR_CR_06		

Table 38 - System Requirement SR_FR_04

Non-Functional Requirements

Balance vector			SR_NFR _01
Priority	Low	Necessity	Low
Stability	Unstable	Verifiability	Medium
Description	The system must balance the camera perpendicular to earth's gravity vector		
Dependencies	UR_CR_01, UR_CR_02, UR_CR_05		

Table 39 - System Requirement SR_NFR_01

Motors			SR_NFR_02
Priority	High	Necessity	High
Stability	Stable	Verifiability	High
Description	The system must use 3 electrically powered motors, one for each axis and it must represent one axis only by being the center of that axis of rotation		
Dependencies	UR_CR_01		

Table 40 - System Requirement SR_NFR_02

Arduino Microcontroller			SR_NFR_03
Priority	High	Necessity	High
Stability	Stable	Verifiability	High
Description	The system must use an Arduino microcontroller to control the entire system		
Dependencies	UR_CR_01, UR_RR_02, UR_RR_04		

Table 41 - System Requirement SR_NFR_03

Movement calculation			SR_NFR_04
Priority	High	Necessity	High
Stability	Stable	Verifiability	Low
Description	Movement must be calculated using electronic sensors connected directly to the Arduino controller via cable		
Dependencies	UR_CR_01, UR_RR_01, UR_RR_02		

Table 42 - System Requirement SR_NFR_04

Rotation limit			SR_NFR_05
Priority	High	Necessity	High
Stability	Stable	Verifiability	High
Description	The system must not be able to rotate in any axis more than 180 degrees from the left end to the right end, except for the Z axis that can rotate 360 degrees.		
Dependencies	UR_RR_02		

Table 43 - System Requirement SR_NFR_05

Device compatibility			SR_NFR_06
Priority	High	Necessity	High
Stability	Stable	Verifiability	High
Description	The system must be able to rotate with a load of at least 165 grams and house a camera of at least 6.35 x 6.35 x 3 cm		
Dependencies	UR_CR_04		

Table 44 - System Requirement SR_NFR_06

Power supply			SR_NFR_07
Priority	Medium	Necessity	High
Stability	Stable	Verifiability	High
Description	The system must use a battery. This battery must be attached to the back of the device and feature a power button with the standard power on/off button used in electronics		
Dependencies	UR_RR_01, UR_RR_02		

Table 45 - System Requirement SR_NFR_07

Enclosed System			SR_NFR_08
Priority	Medium	Necessity	Medium
Stability	Stable	Verifiability	High
Description	Every component except the battery and the motors must be enclosed inside the handle		
Dependencies	UR_RR_01		

Table 46 - System Requirement SR_NFR_08

Structural limitations			SR_NFR_09
Priority	Medium	Necessity	High
Stability	Stable	Verifiability	High
Description	Structure must be composed on a single hollow handle with a diameter not superior than 5cm and it must not weight more than 1kg		
Dependencies	UR_RR_03		

Table 47 - System Requirement SR_NFR_09

System performance			SR_NFR_10
Priority	High	Necessity	High
Stability	Stable	Verifiability	Low
Description	Sensor data turned into motor movement must be completed at least in 30Hz every time		
Dependencies	UR_RR_05		

Table 48 - System Requirement SR_NFR_10

Motor placement			SR_NFR_11
Priority	High	Necessity	High
Stability	Stable	Verifiability	High
Description	The system must use 3 electric motors, one for each axis and each of them must represent only one axis by being the center of that axis of rotation		
Dependencies	UR_CR_01, UR_CR_05		

Table 49 - System Requirement SR_NFR_11

Noise performance			SR_NFR_12
Priority	Low	Necessity	Low
Stability	Unstable	Verifiability	Medium
Description	Motor movement must not surpass 10db of audible noise		
Dependencies	UR_RR_06		

Table 50 - System Requirement SR_NFR_12

Traceability Matrix

Once all the requirements are composed and analyzed, we must ensure that for every User requirement there is at least one System requirement.

To make this analysis, a traceability matrix is used. A traceability matrix has on its vertical axis all the user requirements, and in the horizontal axis the system requirements.

	SR_FR_01	SR_FR_02	SR_FR_03	SR_FR_04	SR_NFR_01	SR_NFR_02	SR_NFR_03	SR_NFR_04	SR_NFR_05	SR_NFR_06	SR_NFR_07	SR_NFR_08	SR_NFR_09	SR_NFR_10	SR_NFR_11	SR_NFR_12
UR_CR_01	X				X	X	X	X							X	
UR_CR_02					X											
UR_CR_03		X														
UR_CR_04										X						
UR_CR_05			X		X										X	
UR_CR_06				X												
UR_RR_01								X			X	X				
UR_RR_02							X	X	X		X					
UR_RR_03													X			
UR_RR_04							X									
UR_RR_05	X													X		
UR_RR_06																X

Table 51 - System requirement traceability matrix

4. Design

This section contains the analysis phase on comparing and selecting the best options for each hardware component as well as the changes that went through while developing the structure of the system.

The section also contains a hardware design section involving all the steps that were made to reach the end hardware design.

The same can be said about the software design section, which talks about the information flow of the system with all the hardware structure already defined.

Hardware Analysis

The system is composed by a set of components communicating with a main controller as can be seen in the figure below.

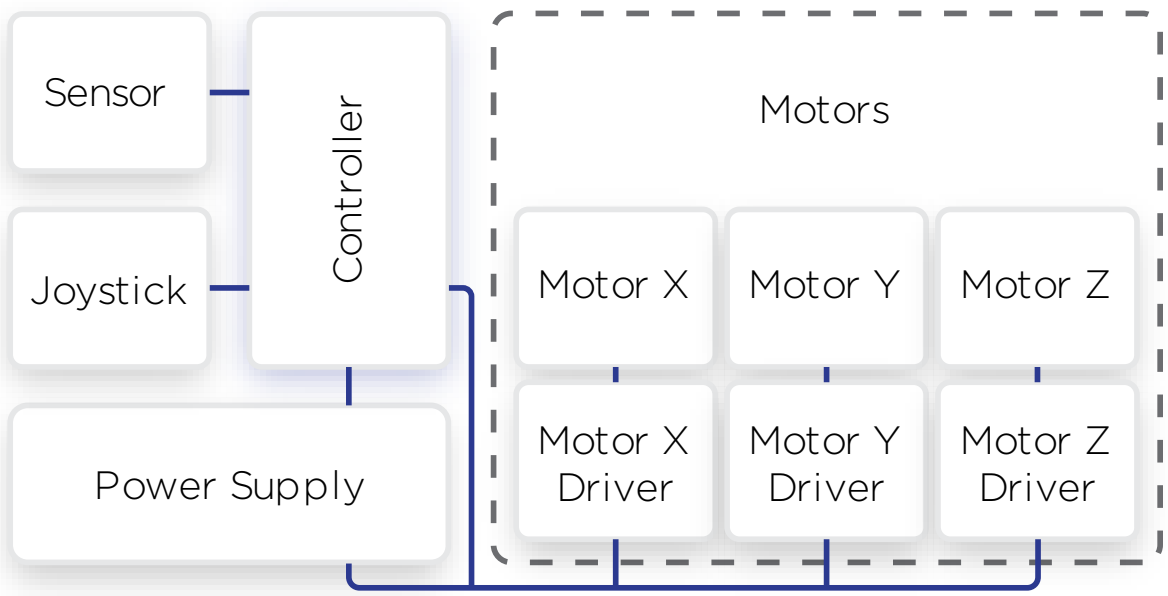


Figure 2 - Design - Hardware components

Controller

The controller board that needed to be used in this project had to be an Arduino, the question remained, which shape of Arduino should I use.

Since the purpose of this project is to try and make it as portable as possible while being fully functional, I needed to search for an Arduino with enough connections that had the smallest size.

From the beginning, motor requirements along with sensors made it impossible to use only one Arduino if it was small or it would need to use a larger Arduino such as an Arduino Mega.

There were two sensors and they needed to be interconnected since it would be using the values of one to counter the margin of error of the other one, so they would need to be communicated.

From this set of requirements, I opted to use two Arduino Nanos since they would have enough connections for two sensors and two motors but kept the smallest size while also including a USB connection for easy coding access. I explored some other options without the USB connection, but debugging was not easy at all specially since I needed to move the device around to debug it. I also tried to use an Arduino Mega, but the amount of unused ports and its size were just too much for the structure of the device.

These Arduinos were connected as a master slave in which one of the Arduinos managed the sensors and communicated the results to the motors. This option proves not to be optimal since the delay of transferring the information safely was too long for the motors to react quickly enough.

Finally, one Arduino Nano was enough for the system to work so for this build, that was the final option.

Sensor

The sensor was a considerable choice on the project both on quantity and on the data that they would provide.

I needed a sensor capable of measuring as an absolute value, degrees of rotation compared to the gravity pull on the place in which we are using the device. For this purpose, there are several options.

- Magnetometer:
    - These sensors can measure angular velocity and speed, as well as directional velocity and speed in 3 axes, this is achieved by using an accelerometer, a gyroscope and a compass that would be able to act in the Z axis.
    - These sensors are a built-in solution that gives absolute values for rotation in all directions.
    - These are also extremely expensive at 40-60€ each.

- Gyroscope:
    - A gyroscope is not capable of measuring absolute degrees of rotation but instead, it is the job of the programmer to determine those values based on the angular acceleration and velocity.
    - For this purpose, there are several options:
        - There are sensors that have a built-in solution for removing the error from the provided data.
        - There are others that do not provide such solution and must be implemented by the programmer.
        - As a final option there are controllers that provide this function when put in series with the sensor.
    - Since one of the requirements for this project is to not use any external built components, the last option was discarded, and I opted for the second option, although the sensor I used for the project had a built-in solution, but it is not official, and I was not aware of it at the time of purchasing the sensor.
    - This option is very cheap, and it allowed me to have several sensors in case one came non-functional. The average prices for these sensors is 1-3€

In terms of quantity and position there is a difference in terms of error calculation depending on where you position the sensor:

- If the sensor is mounted on the base it does not know where the camera is in any moment, but it will also not be affected by movement of the motors therefore being more precise.
- If the sensor is put on the camera it knows where the camera is at all moments, but while the motors are spinning that movement must be considered and that is an additional error calculation.
- The sensor can also be put on each axis and therefore it would only count for one movement on that axis and then use the rest of axis to add precision to the calculations.
- It can be any of the combinations of the options above.

In this project I initially planned on using one sensor in the base as a main sensor and then use a second sensor mounted in the camera to add precision to the calculations using a small portion of it to know where the camera is.

After determining that both the structure and the Arduino would not be able to run two sensors in the system I opted to use only one sensor mounted at the base of the system

My final decision for this system was to use one MPU-6050 Gyroscope mounted at the base of the system. This decision was made since it was cheap, had a lot of community coverage and it was precise and easy to use.

Motor

The motor was the most complicated matter in this development since controlling them requires a lot of electrical knowledge and physical programming.

There were several options for the motors:

- Brushed/Brushless Motors:
    - This is the best option in terms of quality, these motors are comprised of two sections, isolated from each other, what makes them very smooth, silent and they have great amounts of torque depending on the ratio of height vs width. In my case since I needed more torque than speed I opted for more width than height.
    - The problem with these motors is that they do not include a controller and they need some physical programming since the motor only provides you with 3 cables that must be turned on and off in a sinusoidal way. This means that it cannot be controlled by absolute degrees unless programmed manually. There are external controllers for this kind of motors, but they control revolutions above 1000 rpm, so it is meant for high continuous rotations.
    - These motors require additional electronics since they require more energy than the Arduino can provide without burning.
    - They are also very expensive.

- Stepper Motors:
  - This kind of motors have a dented magnetic disc inside that can rotate in steps, usually about 1.8 degrees per step, it also gives the possibility of using half-steps granting you up to 1/8 of a normal step but it requires manual programming.
  - They are usually fast enough and they have high amounts of torque, but they lack precision for a lot of developments.
  - There are several options for these motors:
    - Geared motor: There are some motors that are too small and to compensate they use gears like the servo motors, and they lack position feedback, so it lacks reliability and precision. They get loud as well because of the gears.
    - Normal motors: Commonly found in 3D printers they are big and heavy but silent and precise, they feature controller feedback, so it knows the position and can correct in the case of a mis-movement.
  - This kind of motors are controlled by absolute degrees of rotations using a controller that is usually included. They might also require external power depending on the size, but the small ones can be operated with 5v which is enough for the Arduino to operate.
  - They are usually in the expensive side.
- Servo Motors:
  - This are the cheapest option, these are motors that are very small usually and they use gears to compensate their lack of torque, which means that they are not that precise nor fast, the system would have worked but it would be very slow and noisy.
  - This kind of motors are controlled by giving an absolute angle and they usually do not require a lot of programming or any external electronics since they consume less than 5v and Arduino natively supports controlling servos.

My first decision for the motor was to use a brushless motor with a controller since I imagined I would be able to control it slowly. After trying to program the controller and getting no results, I moved to trying to use the Arduino as a controller. I used a very good guide [4] on how to physically move these motors using sinusoidal waves on each of the inputs. As far as I could tell, the programming was okay, but the electronics were not good enough, so the behavior was not correct for the motor, I could not get it to move the way I wanted them to. Such electronics would imply building an external battery and adapting the current to the motor specifications, it also implies using the correct timings in the Arduino device to make everything smooth. In addition, movement is not the only issue but precise angular movement in absolute positions is what I needed to do, in the end this part was resolved but the movement was still not precise.

I finally decided to move from using brushless motors to stepper motors. These new motors came with their own controllers that would allow me to move them using absolute degrees of rotation very easily and very compatible with Arduino. They also do not require an external power source, so it made the system more compact and reliable. The issue with these motors is the torque and the speed since I bought geared motors. Luckily, the torque is enough to power the camera that I was using for the development.

Structure

The structure that was required for the system was something that could hold on to 3 motors in 3 different axes and that could hold the entire system in one compact and portable package. It also needs that the structure holding the motors are independent one from another since they need to move.

At first, I opted to design the structure in 3 Dimensions to try and calculate if the structure would be resistant enough, since I intended to 3D print all the pieces and make a very compact system. Finally, I decided that the amount of weight added by the plastic and its structural fidelity made it a sub optimal option for the structure.

Lastly, I decided that a better structure would be to use aluminum for the entire structure. This new design would involve manual handling of the materials since I do not own any tools that would allow me to shape metal in the ways that I intended to.

In addition, I added support structures for the motors and the axis of rotation to release the weight of the structure from the motors themselves.

Initial designs also used a joystick that had to be placed on the space that the user would use to grab the structure.

Power Supply

This part of the design was very dependent on the motor choice since they require external power depending on their type.

In the beginning, since I was using brushless motors, I needed a way of powering both the Arduino, the sensors and the motors.

- The Arduino needs 5v 1A of power to work, it can also work with 3,3V 1A depending on the energy that you need to provide.
- The sensor that I chose (MPU 6050) states on its documentation page that it requires 3.3V to operate.
- The motors required 5V and depending on the amperage they would have more, or less torque.

Given these conditions I needed to operate the Arduino in 3.3V or 5V and the motors in 5V but independent from the Arduino connections since they would burn the Arduino if they require more than 1A, which happened.

The chosen power source then was a custom-made battery that could provide 5V and divide that power into the Arduino and the motors since the Arduino would be powering the sensor through its 3.3V port. This battery had plenty of amperage for the motors to have enough torque to move a heavy load such as the camera.

As a second instance, when brushless motors were not an option anymore, but steppers were, they only need 5V and less than 1A to operate so they could be powered using the Arduino. For this reason, it was easier to use a standard external battery supply for phones that operates exactly at 5V 1A and are smaller and more reliable. It also features a USB connection which was the preferred method to connect the Arduino to the power supply since being able to disconnect it made easier the development part.

Hardware Design

This section is a detailed view on the programming and development of the system, the full process including errors encountered and their solutions as well as changes in design made along the way.

This section is divided in the different temporal phases that I used to develop the system, focusing each time on a specific component.

Common concepts

The MPU6050 uses an I^2c connection to transmit data and it has several configuration values to select the type of data you want to receive.

An I^2c connection is a digital protocol that uses 2 cables to transmit data concurrently and other 2 cables to power the device. Being a digital environment, it requires the use of a library to be able to recognize the bits that are being sent, for this, there are several libraries that can be used, each individual iteration will use its own library depending on its needs.

As per documentation specifications [7] the sensor is always configured as follows:

- Gyroscope:
    - This was set to 500 degrees per second, that is, that it is capable of measuring speeds up to 500 degrees per second in which case it will return the maximum value.

- Accelerometer:
    - This was set to return values from -8192 to 8192 for the acceleration. This defines the precision on how to measure gravity and accelerating rotation.

The testing methodology for this phase is to place the sensor in different known positions for the duration of 30 seconds and measure its precision and drift over time.

- ±45 deg X axis
- ±90 deg X axis
- ±45 deg Y axis
- ±90 deg Y axis
- ±90 deg Z axis

The reason why 45-degree rotation is tested in this development is because some sensors do not work correctly with 90 degrees, but they will work well with lower degrees.

The way to follow if the system passes all tests is going to be with the use of a table containing a ● if the test was passed and a ● if it did not

Phase 1: Sensor data acquisition & treatment

Iteration 1: Two-sensor system

Configuration and Libraries

The electronics were connected as the following diagram indicates:

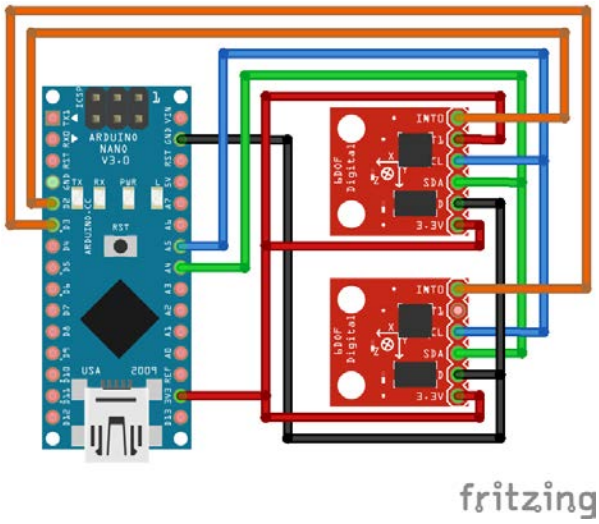


Figure 3 - Sensor development, Iteration 1, Schematics

For this kind of communication to work, an I^2C connection needs to be used as master-slave, being the Arduino the master controller and the two sensors the slaves of the system.

An I^2C connection is used instead of a SPI because of the amount of connections required to connect two sensors to one master. Since Arduino Nano boards have a very limited amount of connections, this was an important step to consider.

The intention of this architecture is to minimize error by using concurrent values to compare instead of one with the drawbacks of being a more complex system that uses more ports.

To read the values I also used “Wire.h” for reading a specific position in memory that is storing the needed value and storing bit by bit into a local variable. This is done as a two-step process since it does it 8 bits at a time and the returned value is an Integer, which in Arduino accounts for 16 bits.

Testing

Position	Immediate	Over 30 seconds
±45 deg X axis	●	●
±90 deg X axis	●	●
±45 deg Y axis	●	●
±90 deg Y axis	●	●
±90 deg Z axis	●	●

Table 52 - Sensor development, Iteration 1, Test results

With all the components set up, the retrieval of data was unsuccessful, the system returned very different values that proved to be incorrect, thus the iteration was terminated and only one sensor was to be used in the system.

Iteration 2: Basic XY system

Configuration and Libraries

The electronics were connected as the following diagram indicates:

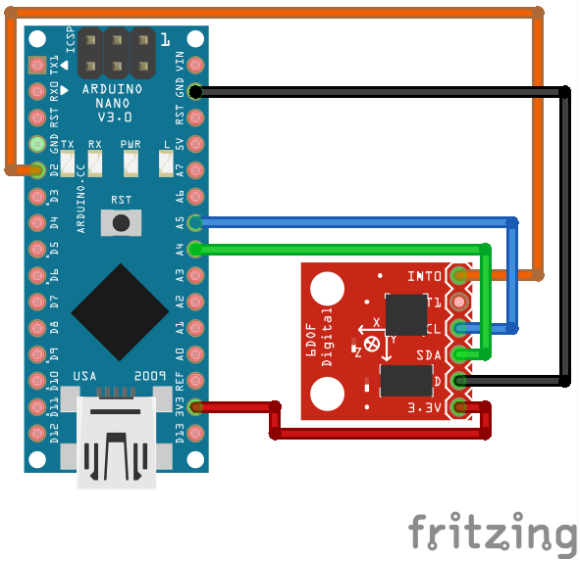


Figure 4 - Sensor development, Iteration 2, Schematics

For this iteration an I^2C interface was used since the available ports are still an issue to consider even with only one sensor.

Wire.h allows to access different sections of the memory on the sensor, that is required, for example, to be able to configure the sensor properly and to ask for specific values stored in its internal memory.

I also used “Wire.h” for reading a specific position in memory that is storing the needed value and storing bit by bit into a local variable. This is done as a two-step process since it does it 8 bits at a time and the returned value is an Integer, which in Arduino accounts for 16 bits.

Data treatment

The first step to treat the data is to calibrate the sensor to calculate what the average values are when stationary, so we can the error of the sensor on each axis. After calculating the error, the next readings will subtract the error to the raw value.

To calibrate the sensor, a loop executes 3000 readings and adds them to a variable that, after completion of the loop, will be divided by 3000. There is a value for each axis of movement and of rotation, and these values are considered the error of each axis, so they are immediately subtracted to every consequent result.

Test

Position	Immediate	Over 30 seconds
±45 deg X axis	●	●
±90 deg X axis	●	●
±45 deg Y axis	●	●
±90 deg Y axis	●	●
±90 deg Z axis	●	●

Table 53 - Sensor development, Iteration 2, Test results

In this case, the X axis was not responding as intended, giving wrong values, a simple solution for this was to change the sensor for another one.

No axis was able to reliable reach 30 seconds every time since the system would stall in the middle of the process and halt the execution of the program. Additionally, the values that would last 30 seconds drifted a considerable amount and became unusable data.

Iteration feedback

For this issue, I tried replacing the Wire.h library for a more specific I2c.h library that would allow me to set the variable I2c.timeout() to 100 ms so whenever it stalled it could continue.

Iteration 3: Final system

Configuration and Libraries

The electronics were connected as the following diagram indicates:

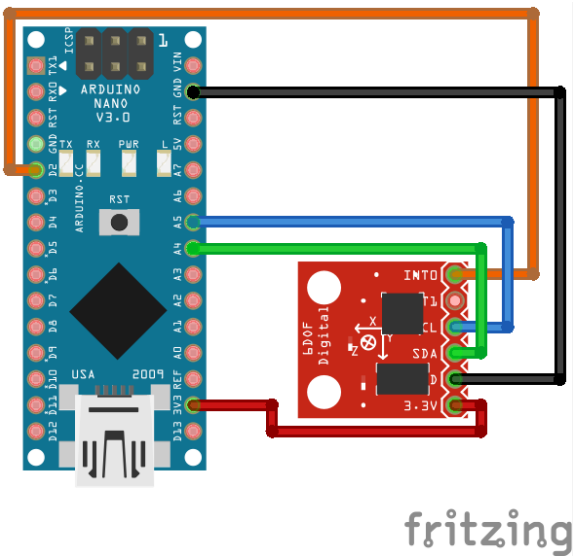


Figure 5 - Sensor development, Iteration 3, Schematics

Instead of using only “Wire.h”, it was necessary to use “Ï2Cdev.h”, a library that can interface with a I^2C environment more efficiently and adding additional features such as time management, which was necessary for this iteration.

Data treatment

The first step to treat the data is to calibrate the sensor to calculate what the average values are when stationary, so we can the error of the sensor on each axis. After calculating the error, the next readings will subtract the error to the raw value.

To calibrate the sensor, a loop executes 2000 readings and adds them to a variable that, after completion of the loop, will be divided by 2000. There is a value for each axis of movement and of rotation, and these values are considered the error of each axis, so they are immediately subtracted to every consequent result.

So far, we have been working with the raw values of the sensor, thus they need to be transformed into angles for a more comprehensive view on their precision. Based on the configuration of the system that was set to measure 500 degrees per second and that the maximum value that it will return based on the documentation [7] is 32750, we can calculate that 1 degree per second accounts for a value of 65.5 so as a first step the following function is applied:

$$\Delta Degree(t) = raw\ sensor/65.5$$

We know from the documentation [6] that the frequency of the system is 250Hz and that is the frequency in which we will be asking for the values, we can then add that value to the equation and then simplify it into a constant:

$$\Delta Degree(250) = raw\ sensor/65.5/250$$

$$\Delta Degree(250) = raw\ sensor/0.262$$

$$\Delta Degree(250) = raw\ sensor * 0.0000611$$

With this new value we can get the angular displacement from one read to the previous one taking into consideration the average error. But there is a problem when rotating the device when not in a flat surface since it is not considering the Z axis, which up to this point we have not calculated, filtered or used since it is not precise enough for our purposes.

To account for this axis, we need to apply the following functions:

$$x = x + y * \sin(z * \frac{\pi}{180deg})$$

$$y = y - x * \sin(z * \frac{\pi}{180deg})$$

$$\frac{\pi}{180deg} = 0.000001066$$

*The reason why we need to use constants instead of the full formula is to do the least amount of calculations possible inside the Arduino itself, so we can achieve all the timing constraints that we have.

With these formulas we have the correct differential of angular displacement for a given time frame of 250Hz including its error. This means that it does not prevent the drift over time since even the slightest error is cumulative and can grow very quickly.

To solve this, we can use the accelerometer since it is usually enough to account for this kind of error. For this we need to calculate the acceleration vector by doing a simple equation:

$$\vec{V} = \sqrt{x^2 + y^2 + z^2}$$

From there we can calculate each angle:

$$x = \text{asin}\left(\frac{x_{raw}}{\vec{V}}\right) * -\frac{1}{\pi/180}$$

$$y = \text{asin}\left(\frac{y_{raw}}{\vec{V}}\right) * \frac{1}{\pi/180}$$

$$\frac{1}{\pi/180} = 57.296$$

To these curated values it is necessary to subtract the calibration values to reduce the error.

Finally, with this set of two values for each axis we can develop a function that will eliminate the drift while maintaining precise values, by trial and error, a value of 99,96% Gyroscope & 0.04% Accelerometer is defined with successful results.

$$Deg_{abs} = G_x * 0.9996 + A_x * 0.0004$$

To dampen the effects of cumulative errors we should implement a function that is also cumulative to reduce such errors, for this, the final value is composed of 90% of the cumulation of previous values and 10% of the new values.

Testing

Position	Immediate	Over 30 seconds
±45 deg X axis	●	●
±90 deg X axis	●	●
±45 deg Y axis	●	●
±90 deg Y axis	●	●
±90 deg Z axis	●	●

Table 54 - Sensor development, Iteration 3, Test results

This version of the system worked as intended.

Phase 2: Motor movement

Iteration 1: Brushless motors

The initial design for this phase was to use a set of 3 brushless motors with an L298 Motor Driver IC. The following schematic shows the electronic components related to this design.

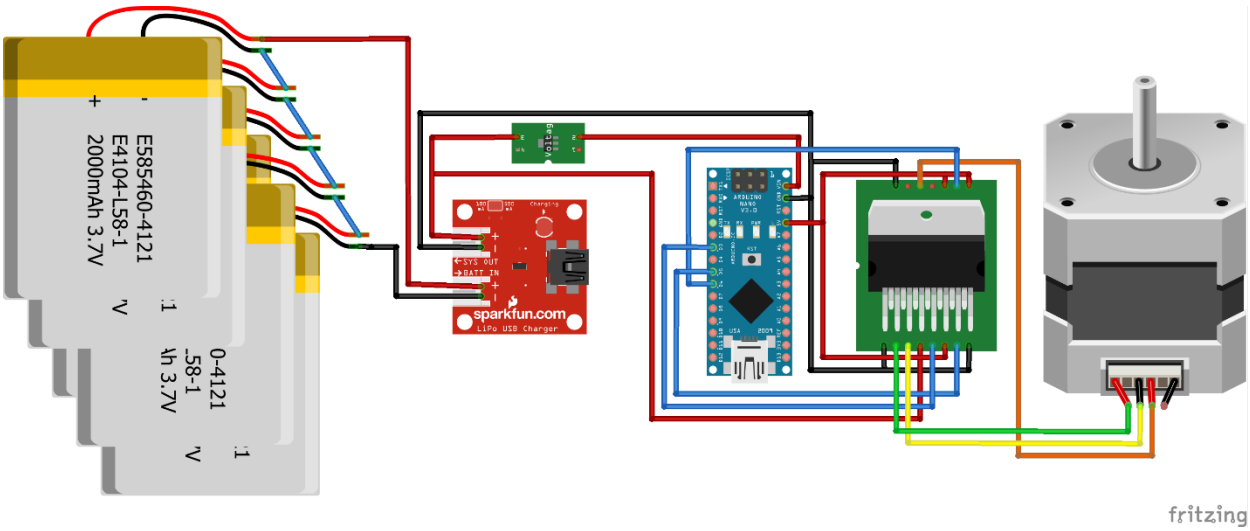


Figure 6 - Motor movement - Brushless schematics

Note that a brushless motor uses 3 cables to operate, since the software used to develop the schematics only had available a stepper motor with 4 wires, it was the one that was used to represent a 3 cabled brushless motor.

The system would use 3 of these motor constructions to manage each axis.

The reason why the system uses an L298 Driver is because the project shall not use any external pre-built components such as an external controller. These drivers allow the microcontroller to create signals to operate the motor fluently.

The way a brushless motor work is using a stator and a rotor. The stator is the part that stays stationary, containing a set of anchored windings on ferrous cogs. The rotor is the rotating shroud with permanent magnets.

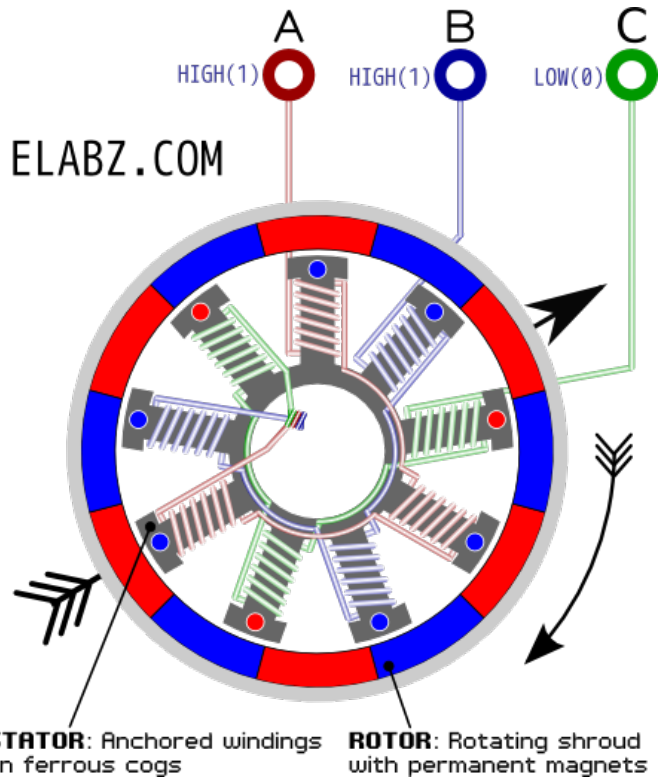


Figure 7 - Motor movement - Brushless motor with 12 magnetic poles and 9 winding cogs [9]

The idea of this motor is to have always 3 points of maximum magnetic force that will move the rotor into the position on which the force is balanced. But instead of maintaining the same 3 points constant, when rotated along the stator, the rotor will follow the motion.

The reason this idea would work is because even though the brushless motor does not communicate to the Arduino to report its position like a stepper motor would do, the Arduino itself is giving the orders on where to move, and, if the system does not skip any cycle, the Arduino controller should always know the current position since it must be stored in a variable anyways.

The reason that we can safely assume that the system will not skip any cycle is that the force that the system can take was already considered in the design phase of the project.

To move the motor, the controller must use a sinusoidal wave function as a pulse width modulation within the clock timer of the controller to operate the three points of maximum magnetic force.

To ease the task of calculating the sinusoidal function at every angle of all three cables per each motor, a pre-built array is used containing all the necessary values to fulfill a full rotation. To better understand the operation and the timing that the clock must undergo, the following diagram represents these concepts graphically.

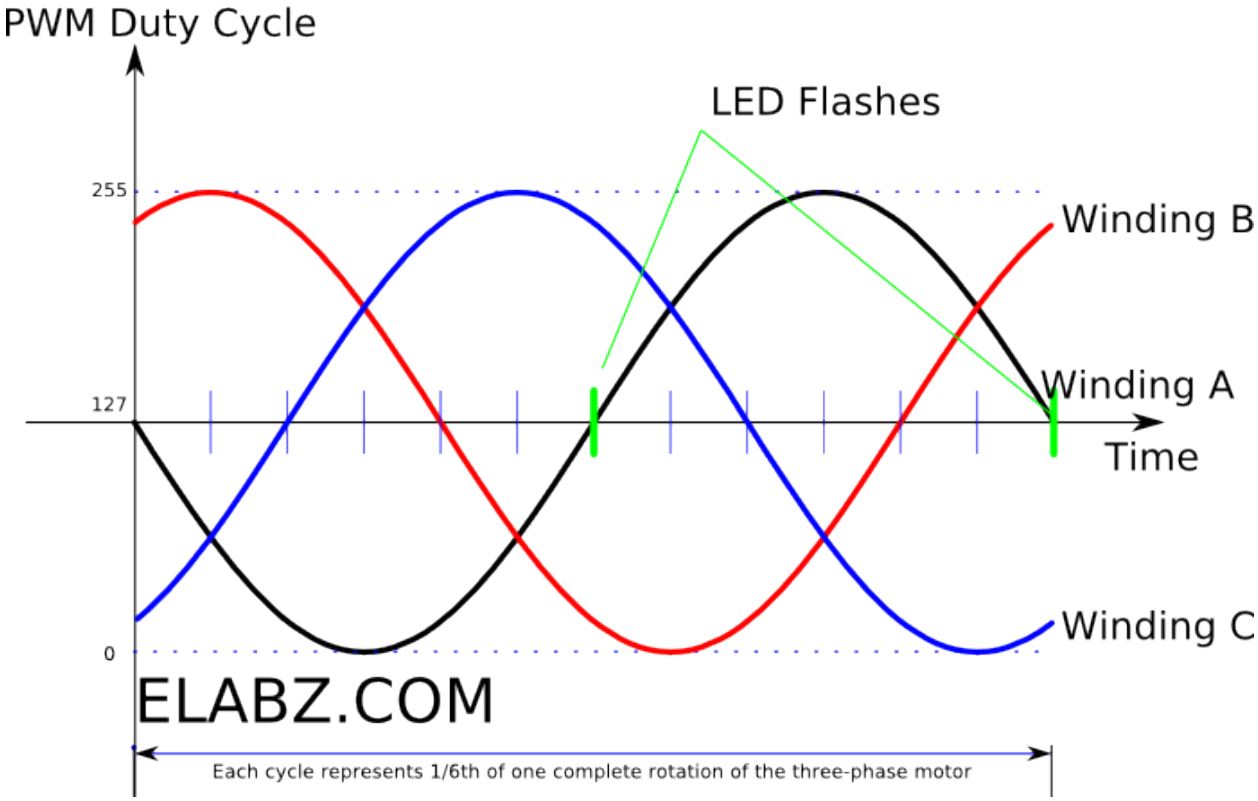


Figure 8 - Motor movement - Brushless controller PWM timer [10]

Since the clock timers are being used for each motor, the timers TCCR0B, TCCR1B and TCCR2B were used. Therefore, this microcontroller could only handle up to three motors.

The easiest way to achieve movement within the system is to directly set the clocks to the sinusoidal array previously described with a bias of 5 values from a cable to another one. This method allows for both forward and reversed rotations, but it did not allow for precise angular rotations where a degree is provided and then the motor moved into the position. To solve that issue, another array containing the original degrees correlated to the sinusoidal array was used, so whenever a degree was provided, the system would move the system until it reached that position.

This method was a very simple way of making the system run, but rotation was not precise nor smooth, and the system failed to maintain a constant direction.

At first this issue was attributed to the battery failing, which indeed proved to be correct since the battery controller was broken and it was not supplying enough voltage, thus the correction was to remove the battery controller temporarily to check if more voltage would solve the issue. In return, the motor driver burnt along with the Arduino and still the motor was not able to maintain a constant direction.

The second thought to resolve the issue was to use a code provided by ArduinoDeXXX in his intent to replicate brushless motor movement in the same situation as this project [11], with this new tested code, the system still did not perform any different from before, leaving only the decision to move to a different motor technology that would suit the system better.

Iteration 2: Stepper motors

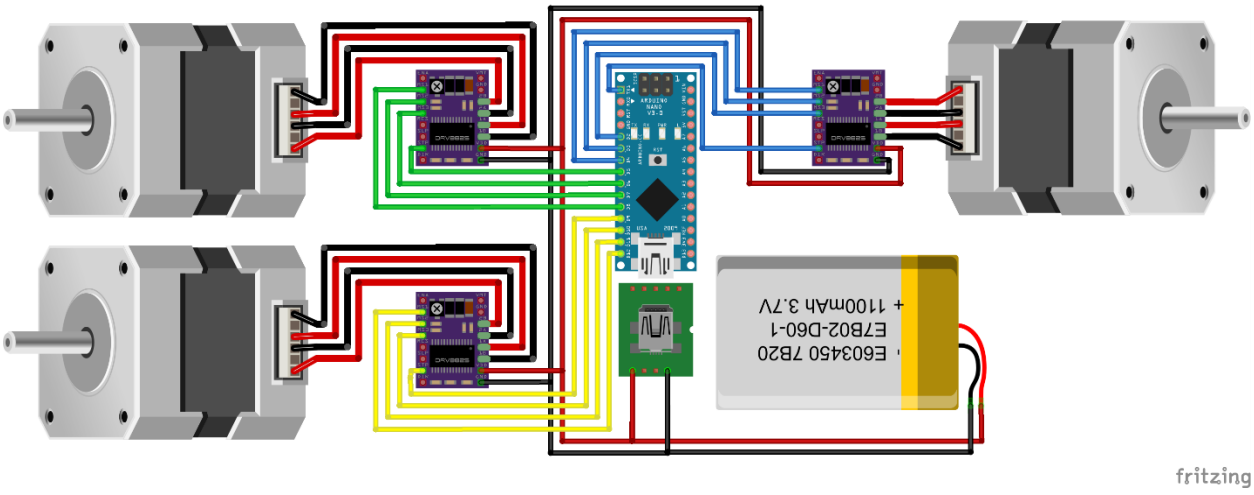


Figure 9 - Motor movement - Stepper motors schematic

The reasoning behind using these motors is discussed in the design section of this document but shall be repeated for understanding purposes. These motors can operate by degrees of rotation instead of by constant revolutions and they feature motor communication to send the current position value to the microcontroller to ensure correct movement even if the motor skips a step.

These motors require the use of an external controller to operate since they offer a digital connection instead of an analogic one.

The way these motors work is like the brushless counterpart but with a slight change on the structure. They are composed by a stator and a rotor. The stator is the static component that is composed by a set of equidistant magnets while the rotor is the moving section which is composed by a set of biased cogs that are separated by the same distance as the stator but are biased one from another. A diagram will help to understand how this system works.

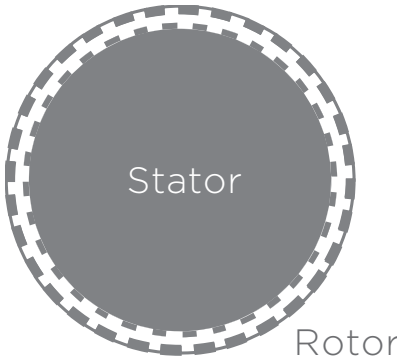


Figure 10 - Motor movement - Stepper motor

The stator itself has several cogs biased one with each other, so instead of using several anchored windings to define the phases, it uses several rows of rotors with a bias defined by the number of phases that the motor has, for example a 3 phased stepper motor has the following stator structure.

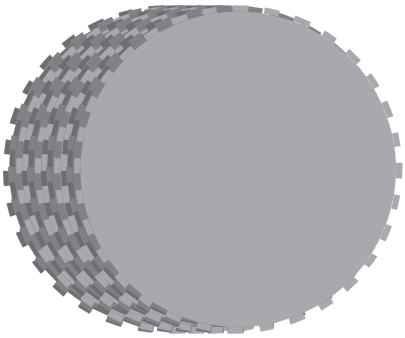


Figure 11 - Motor movement - Stepper motor 2

Where each colored cog represents one phase.

The software side of things would work similarly to the brushless motor with the exception that these motors have 4 or 6 cables depending on the phases and some additional data cables to communicate with the microcontroller. This means that the system sends a signal to each cog depending on which one will have the maximum force of movement. Each movement is measured on steps since whenever a cog finishes moving the rotor, the system reliably stops in that position.

A concept consequence of this way of working is that the steps are already defined degrees of rotation, mostly set to 1,8 degrees per step, which is not very precise. To counteract this lack of precision, micro-stepping comes into play, which treats each cog as a component of a complex system that involves using all the cogs at the same time with different values to allow for a step that is as precise as the original steps divided by the number of phases of the system.

Luckily, this motors already have an official library for Arduino called AccelStepper.h that was used to develop this system. This library includes already micro-stepping and communication with the motors. It also does not use PWM, so the microcontroller is able to use as many as the ports can handle.

To move the motors the only need that was needed was to set the desired degree and call the AccelStepper.h function called runSpeedToPosition() that will move the motor only one step, micro-stepping considered. This was done to prevent the motors to move to the final position and risking having a variable time in the scheduling of the system since this is a real time system. This also means that the function has to be called as much as possible for the motor to have any perceivable performance.

Phase 3: Structure

Power Supply

As described in the requirements, the system needs to work with an independent power supply, in this case in the form of a battery.

The system is composed by 2 different power requirements, one 3.3v source to power the Arduino and the sensors and the motors that would work at 5v. Given the separation of power requirements, a custom battery was made using 5x 3.7V Li-Po batteries of 3400mAh, after some research [5], the best outcome was to use them all in series to get the maximum amperage and battery life, with the addition of a 3.7V to 5V converter to separate the power channels and a battery controller to charge them using a USB cable. This last component is very necessary when using a Li-Po battery system, since they require careful attention and maintenance since they explode very easily, luckily, this controller oversees managing all those problems.

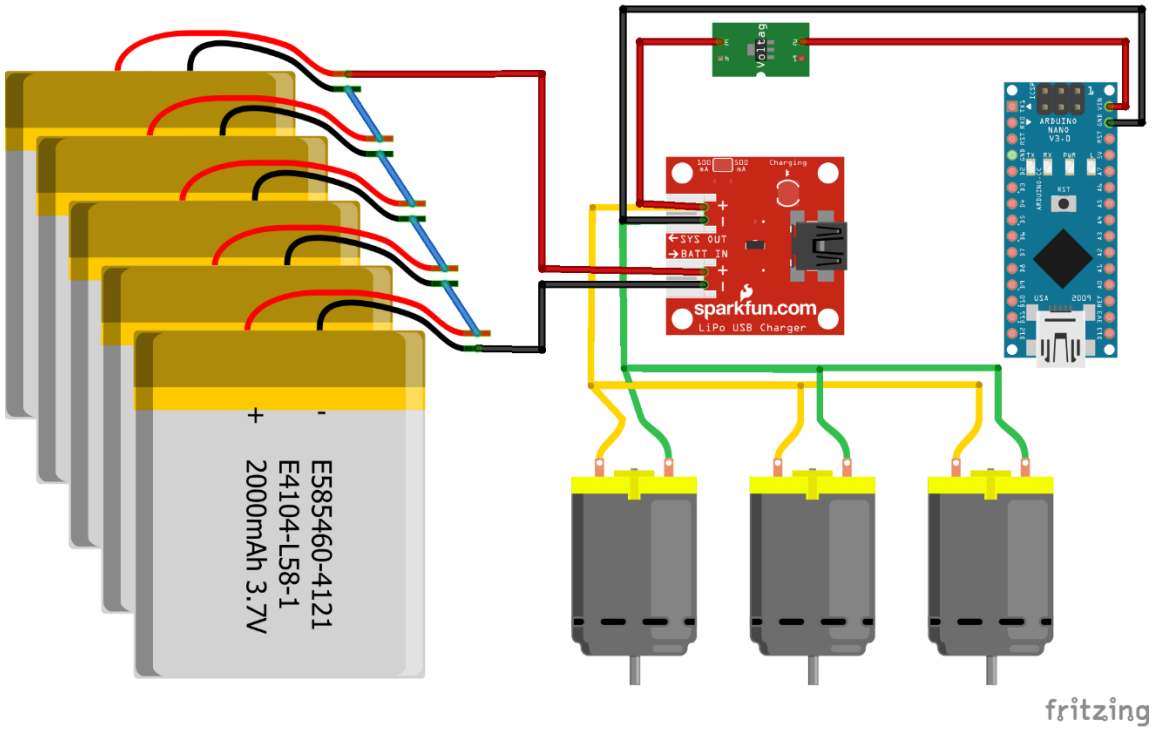


Figure 12 - Structure development - Custom Li-Po battery diagram

This system worked but presented a lot of failure points that ended up breaking the system. A different, more enclosed solution was to find an external battery power bank and use the standard USB to power the Arduino, and since Stepper motors require 5V with little amperage, it was enough to power the entire system without much failure points. This is given the fact that the system changed for stepper motors in the last iteration of the motor movement.

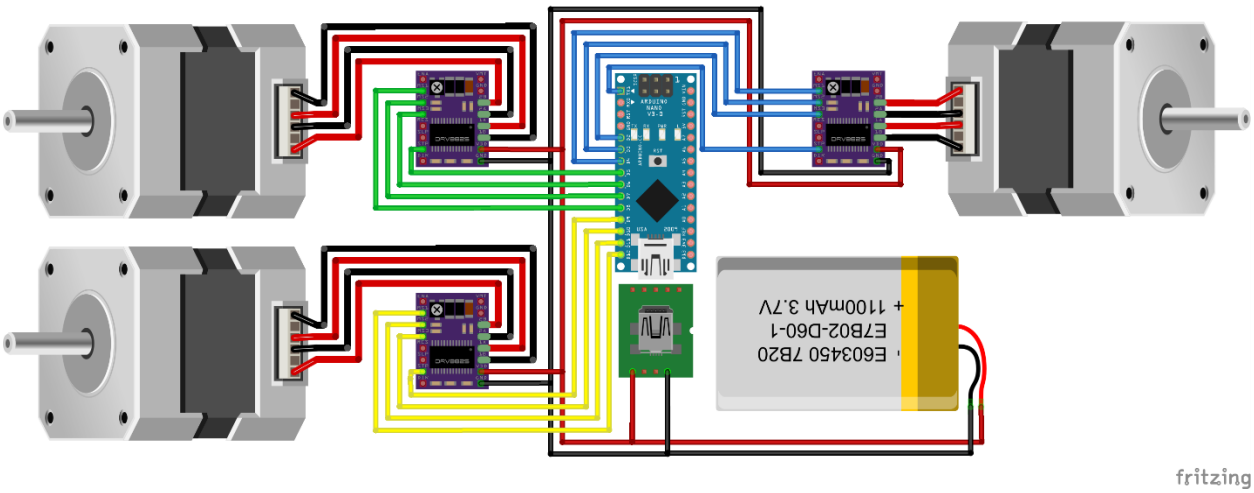


Figure 13 - Structure development - Aftermarket Li-Po battery diagram

Note that even though the Arduino should be capable of handling the load of the motors, it is still better to let the battery itself handle that load, thus the motors are connected directly to the battery. Additionally, the battery on the diagram is only symbolic as well as the motors and motor controllers, since the software used to create the diagram does not include the exact models used in the project.

Joystick

The use of a joystick was necessary to add the pan functionality to the Z axis, it was wired in the following way:

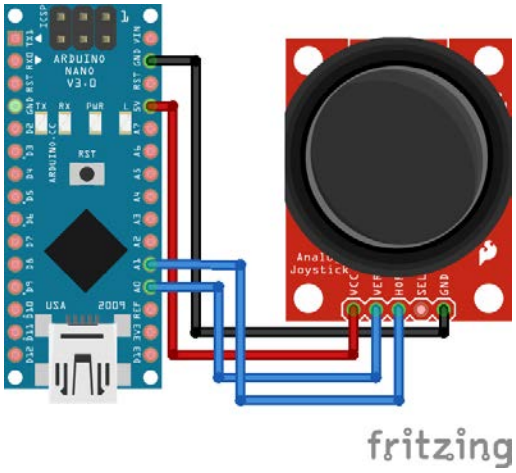


Figure 14 - Structure development - Joystick schematics

The use of the joystick in the system is simple, the value that it returns is parsed from 0-1024 into values regarding absolute positioning. Since the Z axis does not have a rotation limit, it rotates 360 degrees from -2048 to 2048, these values being 0 degrees and 360 degrees on the stepper motors.

Software Design

The basic structure of the software system is represented in the following UML diagram, describing the functions and data types that are included within the different parts of the software controller.

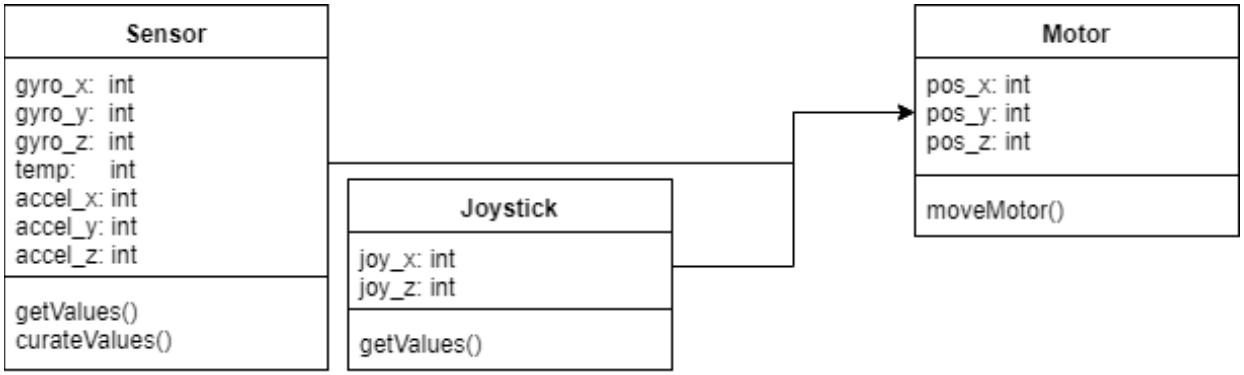


Figure 15 - Software Architecture

Flow Diagrams

This section contains the information flow of each of the software components with a brief description of each one of the flow diagrams.

Calibration

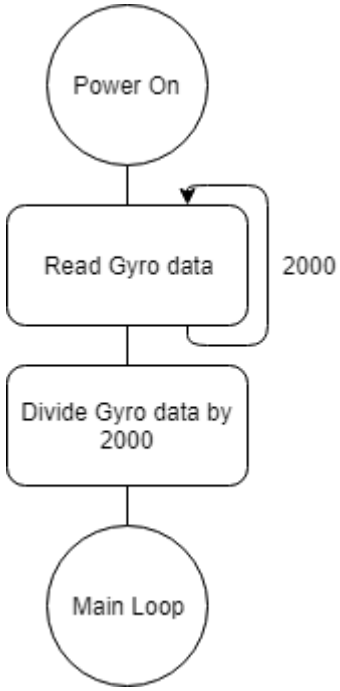


Figure 16 - Flow Diagram - Calibration

This diagram shows what happens when the device is first initiated, that is, the calibration process that the system must undergo before the execution of the main loop of the system.

This calibration consists on reading data from the sensor 2000 times and then dividing the addition of those 2000 values between 2000 to get the mean value of all the executions.

Note that this section of code is only executed once at the startup of the system and the values are used in every iteration of the values calculation on the main loop, but it is not represented in any diagram.

Main loop

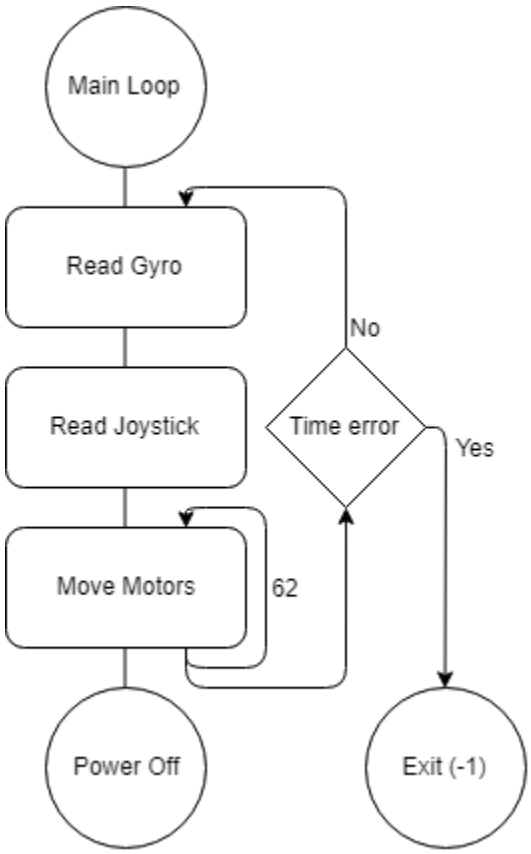


Figure 17 - Flow Diagram - Main Loop

This diagram shows what happens in the main loop of the system, after initialization and calibration has completed.

The main loop consists of 3 functions, Read Gyro, Read Joystick and Move Motor. The first two functions are executed normally and store their values on public variables while the move motor function is executed 62 times to use as much time as possible in that function, so the system performs as expected.

The loop always checks if the execution of all the tasks takes less than 4ms, if it does, the system continues executing the loop normally. If, on the contrary, it does not take less than 4ms, the system throws an error in the serial error channel and the system halts.

The loop will continue its execution until the system is powered off or an error is thrown.

Sensor

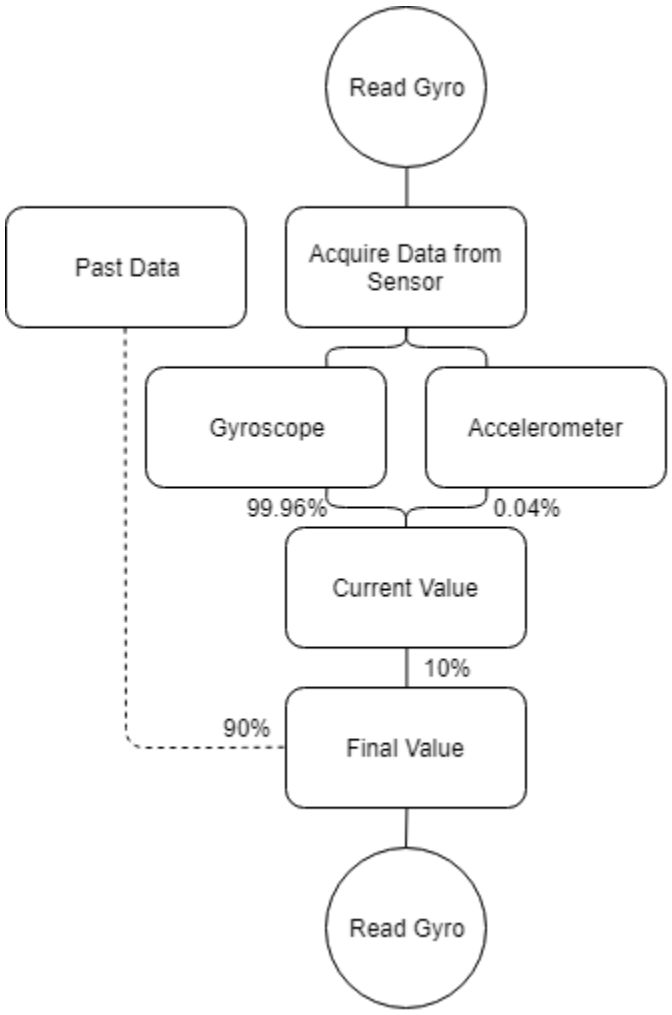


Figure 18 - Flow Diagram – Sensor

The sensor function is called read gyro and it is part of the main loop of the system. It starts by acquiring the raw data from the sensor, then it translates the values into user readable degrees and it merges into a current value using most of the values from the gyroscope and some values from the accelerometer to counteract drift.

Finally, the current value is merged with an accumulation of past values to further mitigate unwanted errors in the calculations and thus counteracting any remaining drift that can escape from the previous calculations.

Joystick

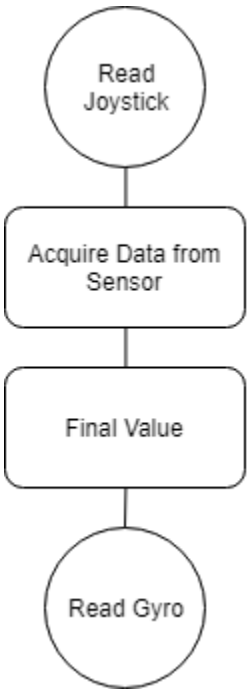


Figure 19 - Flow Diagram – Joystick

This is a very simple function that oversees getting the raw values from the joystick that is present on the system and storing those values on public variables for their posterior use in the move motor function.

Move Motor

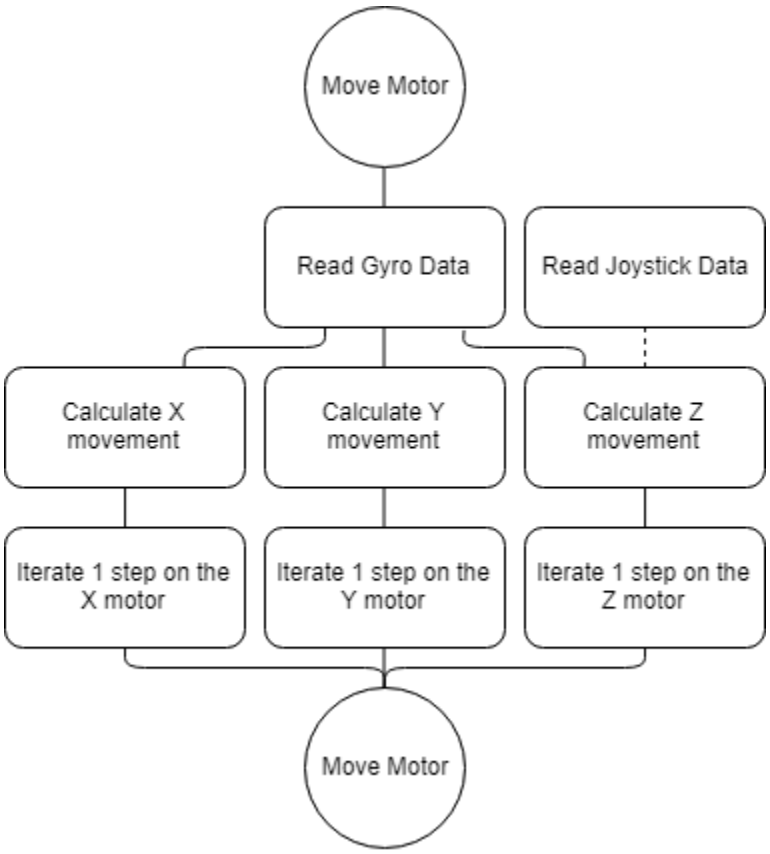


Figure 20 - Flow Diagram – Move Motor

This is the function that oversees adapting all the values to be used in the motor movement. That means adapting the values gotten from calling the read gyro function earlier in the main loop for them to be mapped into the usable value range of the motor, this is done for each motor independently since each motor is controlled independently.

The Z motor is different however, since it uses data from both the sensor and the gyroscope to operate so both values must be considered in the same amount.

5. Development & Testing

This section contains the development of both the structural components and the software. Having the design section complete, it is a matter of putting together all the concepts to make the full system.

Hardware Development

Electronic assembly

Once all the components have been designed, the hardware junction of all the parts looks like the following diagram:

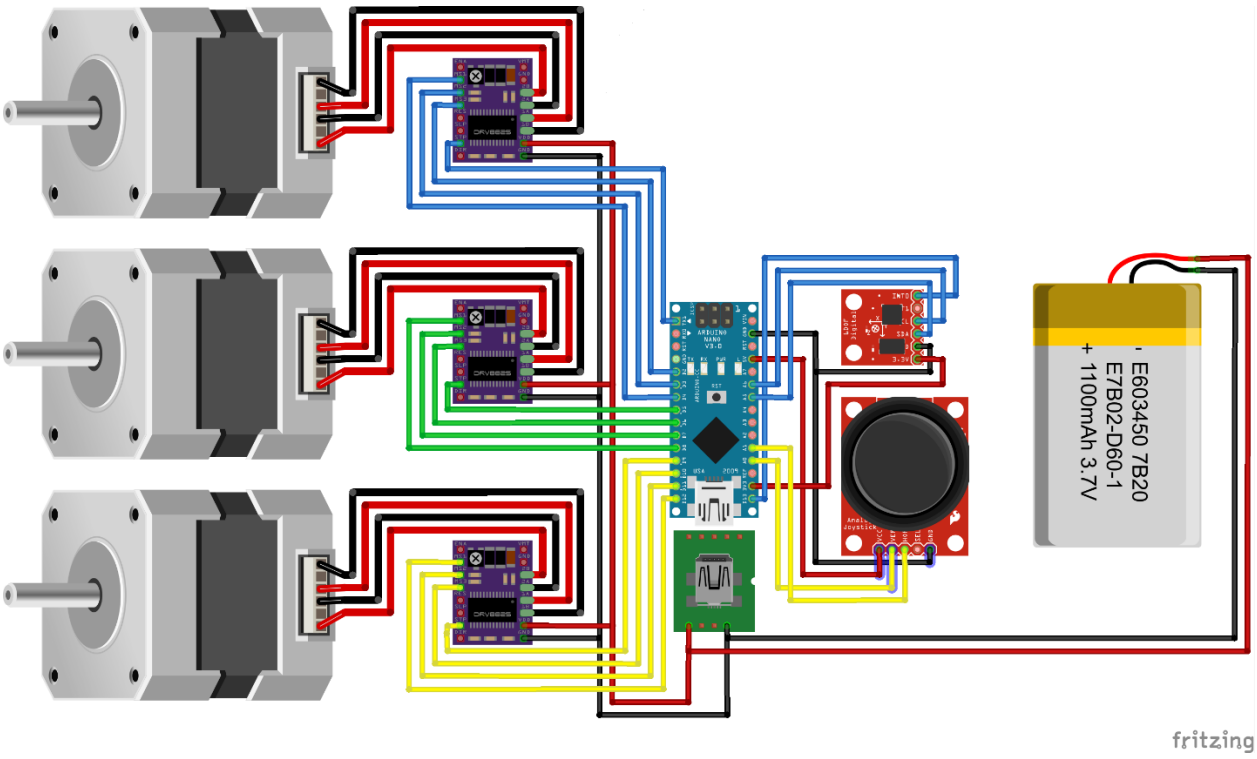
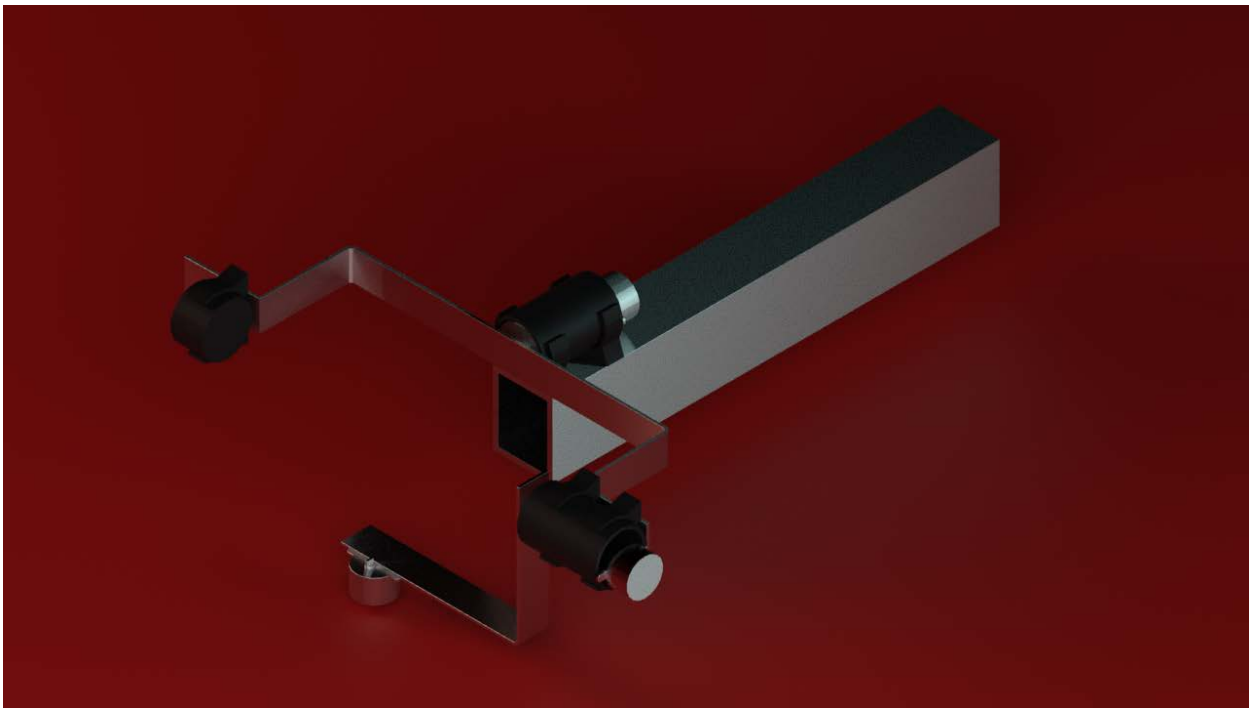


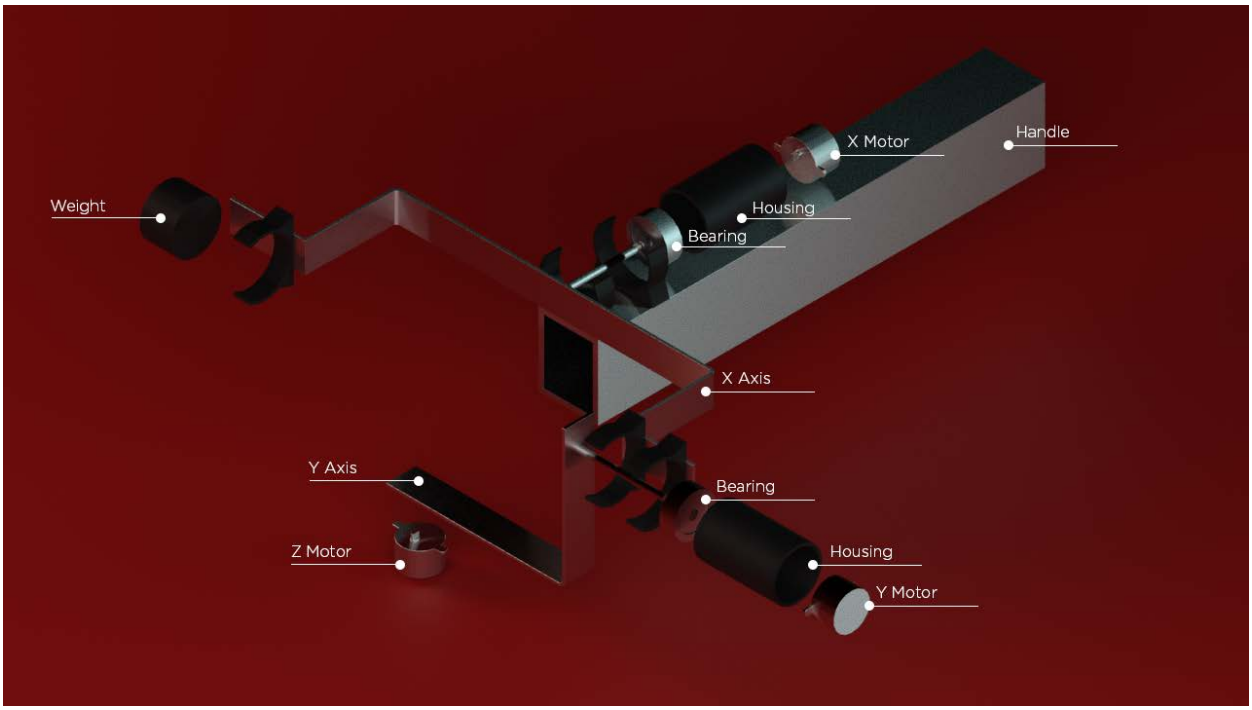
Figure 21 - Structure development - Full System Schematics

Physical structure

The finished system looks as follows:



Picture 18 - Structure development - structure



Picture 19 - Structure development - decomposed structure

Both the controller and the sensor are inside the Handle, while each motor corresponds to one axis, described in the picture above. Each motor needs a housing to release the stress of the components into the bearings instead of into the motors themselves since they do not have enough power to handle that kind of stress.

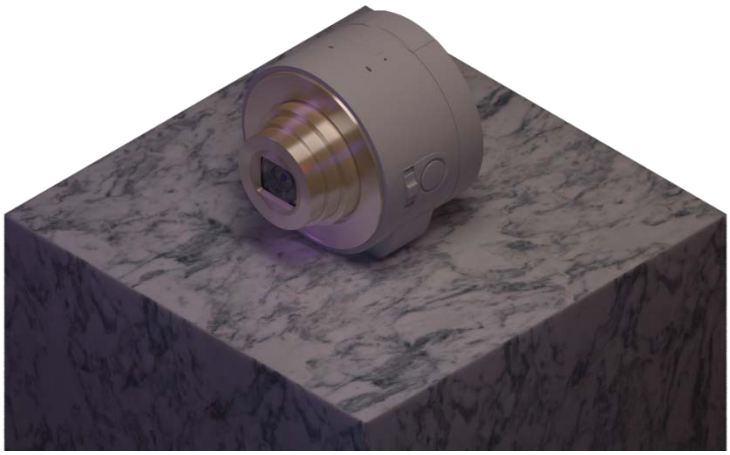
X axis has a weight on the other side to make the system more stable and to add the functionality of self-stabilization. They are all movable pieces, so they can be calibrated for every use.

Y axis does not need a counterweight since it is working with gravity instead of against it. That motor still uses a housing to release the stress to the bearing.

Z axis is simply attached to the Y axis since the stress of rotating a camera is not too high for the motor to handle and adding more weight to the system would decrease the performance of other axes.

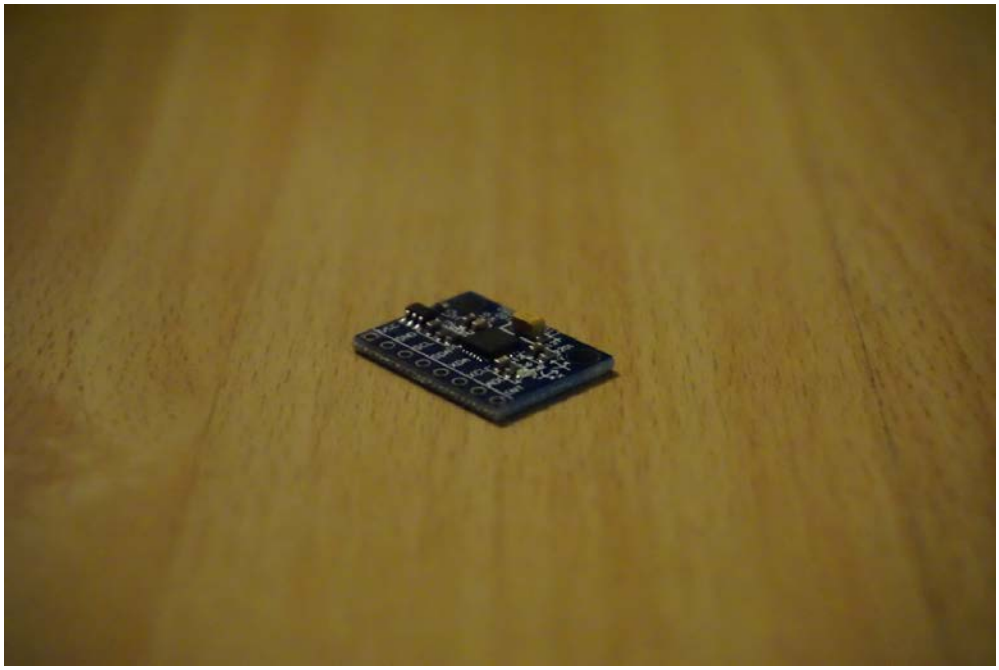
All the structural pieces are made of aluminum and manually bent using heat. The structure is strong enough for daily use and the only freedom of movement comes from the motors themselves and account for ±3 degrees on each axis.

The camera to be used and to which the system is calibrated is the Sony QX10:



Picture 20 - Structure development – Camera

The sensor used as a gyroscope and accelerometer is the Invensense MPU6050

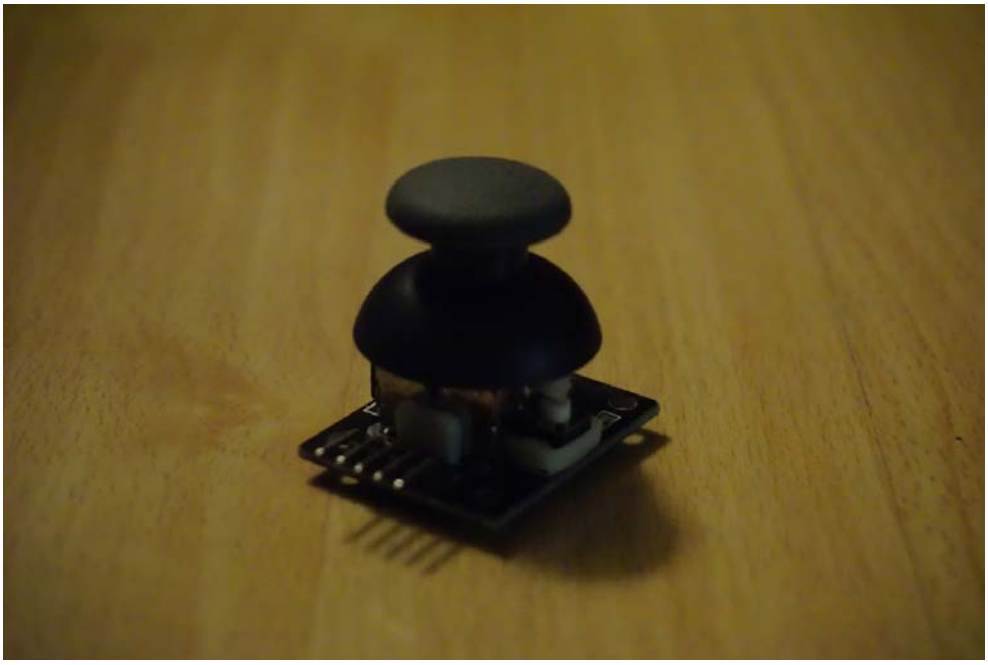


Picture 21 - Sensor

The microcontroller to be used is an aftermarket Arduino Nano that does not have support for variable voltage control.

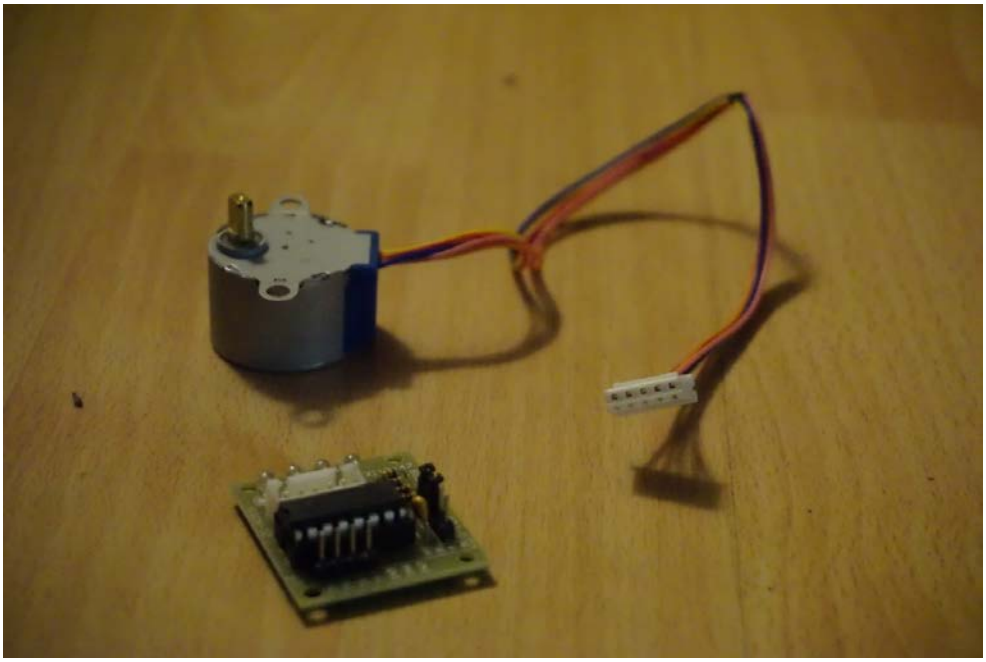
Picture 22 – Microcontroller

The Joystick used for the project is an analog sensor called KY-023



Picture 23 - Joystick

The motors used in the project are a set of 5V stepper motors with their drivers, the specific model is 28BYJ-48 for the motors and ULN2003 for the drivers



Picture 24 - Motor

Software Development

With all the components on their final versions it is time to create a single controller containing all of them. Since this is a real time system, the components must be integrated as a part of a real time system.

The timing for all function is going to be described in the following table

Function	Average (ms)	Max Value (ms)	Sample Size
read_gyro()	2.318917 ms	2.397552 ms	39.000
move_motor()	0.021950 ms	0.021970 ms	411.000
read_joy()	0.224000 ms	0.224200 ms	221.000

Table 55 - Function timings

For a more detailed description on each function see the information below:

- read_gyro: This function oversees the sensor component, by communicating with the sensor to receive data and then cleaning it. Values are stored in global variables.
- move_motor: This function only performs one step of movement for each motor each time it is called, so reaction times must be considered.
- read_joy: This function oversees the communication with the joystick, values are stored in global variables.

There are other sections of the code in charge of calibration and initialization of the components, but they are not considered since there is a calibration phase every time the device is turned on and it is an event that only occurs once.

Per requisites the system must perform with at least 30Hz, per configuration of the sensor, the requests are made at 250Hz. From these two values we will consider the most restrictive one as the one to follow, thus the main cycle will be set up to be 250Hz or 4ms

Additionally, no function has a strict dependency with the other functions, the only one that exists is between the movement of the motor and the acquisition of data, thus move_motor depends on both read_gyro and read_joy but if it lacks that information the motors simply will not move, execution of the program can continue.

With all this information, the real time system can be built as described below.

Function	Computing (ms)	Deadline
read_gyro	2,397552	4
read_joy	0,224200	4
move_motor	0,021970	4

mc	4,000000
sc	4,000000

Table 56 - Scheduling data

Scheduling			
4ms			
read_gyro	read_joy	move_motor x62	
2,397552	0,021970	1,362140	
3,983892			0,016108

Table 57 - Scheduling

As seen in the previous table, the function move_motor is executed 62 times as it is the maximum amount of times that it can be executed in the worst-case scenario without reaching the deadline. The implementation of this function uses a loop that executes 62 times.

The numbers displayed at the bottom of the scheduling table represent the maximum time that the system would take in the worst-case scenario.

Testing

In this section the system is tested on all its components to verify that every requirement is met at each final state of each phase.

The structure to describe each test is a table containing the following fields:

- **Name**: Name of the test to be done. The name must be descriptive and must summarize the content of the test.
- **ID**: Identifier of the test. It follows a format FT_XX where FT denotes Functional Test and XX is an incremental padded number to identify the test.
- **System**: System over which the test is performed, in this case it refers to the Sub Phases of the development section of this project
- **Result**: Specifies if the test concluded correctly or incorrectly, its possible values are “valid” or “invalid”
- **Description**: Description of the objective for doing the test
- **Actions**: Actions to be performed to complete the test

	Sensor rotation		FT_01
System	Sensor	Result	Valid
Description	The sensor must work on its X and Y axis considering the values from the Z axis. They must work in -90 deg to 90 deg. It also needs to be stable and not drift the values over time when stationary.		
Actions	• Place the sensor on a stable surface for 60 seconds and verify that the values are all 0 and they do not drift over time. • Place the sensor on a 45-degree surface on the X axis for 60 seconds and then rotate the sensor. The new value should be 45 degrees on the Y axis and 0 on the X axis. No values should drift when the sensor is stationary. • Place the sensor at -90 degrees on the X axis and then rotate it to 90 degrees. Values must return -90 and 90 respectively and no drift should happen when stationary in any of these positions • Place the sensor at -90 degrees on the Y axis and then rotate it to 90 degrees. Values must return -90 and 90 respectively and no drift should happen when stationary in any of these positions		

Table 58 - Test FT_01

	Sensor rotation Z axis		FT_02
System	Sensor	Result	Valid
Description	The sensor must return correct Z axis values without considering X and Y values.		
Actions	• Place the sensor on a stable surface for 60 seconds and verify that the values are all 0 and they do not drift over time. • Rotate the sensor 180 degrees for 60 seconds and measure the value. It should be 180 and it must not drift over the period of 60 seconds		

Table 59 - Test FT_02

	Sensor calibration		FT_03
System	Sensor	Result	Valid
Description	Sensor calibration must be sufficient to counteract initial noise that might be caused by the sensor itself		
Actions	• Measure the values of the sensor without calibration • Compare these values with the result of the calculated error values for calibration • Both values must be similar with an error of ±15%		

Table 60 - Test FT_03

	Joystick		FT_04
System	Joystick	Result	Valid
Description	Joystick must return correct values and the system must perform the correct tasks when the joystick is used		
Actions	• Move the joystick to the left-most position and verify that the Z axis motor moves to position -180 • Move the joystick to the right-most position and verify that the Z axis motor moves to position 180 • Press the Joystick and verify that the Z axis motor returns to position 0		

Table 61 - Test FT_04

	Motor movement		FT_05
System	Motor	Result	Valid
Description	The motors must move in the correct orientation for each axis.		
Actions	• Move the system ±45 degrees on the X axis and verify that the camera is balanced at 0 degrees • Move the system ±45 degrees on the Y axis and verify that the camera is balanced at 0 degrees • Move the system ±45 degrees on the Z axis and verify that the camera is balanced at 0 degrees		

Table 62 - Test FT_05

Motor noise			FT_06
System	Motor	Result	Valid
Description	The motors must not surpass the noise limit set in the requirement section		
Actions	• Move the system arbitrarily on any axis and measure its performance with a noise detection system. In this case a mobile phone microphone can be used. • The detected value must be below the noise limit set by the requirements on this document		

Table 63 - Test FT_06

Motor limit of rotation			FT_07
System	Motor	Result	Valid
Description	The motors must not surpass the limit of rotation on each axis		
Actions	• Move the system ±180 degrees on the X axis and verify that the camera is balanced at ±90 degrees respectively • Move the system ±180 degrees on the Y axis and verify that the camera is balanced at ±90 degrees respectively		

Table 64 - Test FT_07

Passive Stabilization			FT_08
System	Structure	Result	Valid
Description	The system must stabilize the camera on the X and Y axis with a free movement less than 5 degree.		
Actions	• Disconnect the power to the system • Move the system ±45 degrees on the X axis and verify that the camera is balanced at 0 degrees with an error of ±10 degrees • Move the system ±45 degrees on the Y axis and verify that the camera is balanced at 0 degrees with an error of ±10 degrees		

Table 65 - Test FT_08

Arduino certified controller			FT_09
System	Controller	Result	Valid
Description	The system must be using an Arduino controller, so this should be tested to verify that the software can be run		
Actions	• Connect the system via USB into a computer using the USB cable at the back of the handle. • Open the Arduino Software and verify that there is an Arduino Nano and has an assigned COM port.		

Table 66 - Test FT_09

Camera compatibility			FT_10
System	System	Result	Valid
Description	The system must work with the intended camera		
Actions	• Place the camera on the Z axis camera connection. It should fit without any issues. • Ensure that there is no power on the system and measure the balance of the system with the camera mounted. • With the system powered on, move the Z axis and ensure that the motor can handle the weight of the camera. • Move the X axis and the Y axis to -45 degrees and ensure that the motors can handle the weight of the system by ensuring that the camera stays balanced at 0 degrees in both axis.		

Table 67 - Test FT_10

Power supply			FT_11
System	Power supply	Result	Valid
Description	The system should work with a battery as a power supply and it should meet the requirements of this project		
Actions	• Ensure that the battery is located at the back of the handle and that it contains a standard power on/off button • Press the power button and verify that the system turns on by looking at the red lights at the front of the handle • Press the power button again and verify that all the lights turn off		

Table 68 - Test FT_11

Feedback			FT_12
System	System	Result	Valid
Description	All the states of the LEDs on the system must work as intended		
Actions	• Power on the device and verify that there are 15 red LEDs at the front of the handle turned on. • Wait for 10 seconds and verify that the red LEDs are blinking • Power off the device and verify that no LEDs remain on		

Table 69 - Test FT_12

	Portability		FT_13
System	System	Result	Valid
Description	The system must not have any external wired or wireless connection and it must be self-powered		
Actions	• Move more than 100 meters away from the starting test environment and verify that the system can still perform under its own power and data acquisition. • With the system powered on move the X axis -45 degrees and verify that the camera is balanced at 0 degrees		

Table 70 - Test FT_13

	One handed use		FT_14
System	System	Result	Valid
Description	The system must be small and light enough to be handled by one hand		
Actions	• Grab the device with one hand and try to rotate 90 degrees on any axis. The user should be able to perform this task with one hand and without the need to change its hand placement during the test • Place the device on a scale and verify that the weight is not superior than 1kg • Measure the diameter of the handle and verify that it does not surpass 5 cm		

Table 71 - Test FT_14

	System Performance		FT_15
System	System	Result	Valid
Description	The system must move the motors at least 30 times per second		
Actions	• With a camera, record at 30fps the X axis red LEDs at the front of the handle, these LEDs change every time the motor moves. • Move the system on the X axis arbitrarily during 1 second or more while the camera is recording. • On the video recorded verify that each frame has a different LED combination than the previous frame.		

Table 72 - Test FT_15

Traceability Matrix

We must ensure that all the tests fulfill all the requirements. To ensure this, a set of traceability matrix are done to represent which test is overseeing which requirement.

The traceability matrices are divided in Functional Requirements and Non-Functional Requirements:

Functional Requirements Traceability Matrix

	FT_01	FT_02	FT_03	FT_04	FT_05	FT_06	FT_07	FT_08	FT_09	FT_10	FT_11	FT_12	FT_13	FT_14	FT_15
SR_FR_01					X										
SR_FR_02		X		X											
SR_FR_03									X						
SR_FR_04												X			

Table 73 - Testing traceability matrix – Functional Requirements

Non-Functional Requirements Traceability Matrix

	FT_01	FT_02	FT_03	FT_04	FT_05	FT_06	FT_07	FT_08	FT_09	FT_10	FT_11	FT_12	FT_13	FT_14	FT_15
SR_NFR_01	X		X		X			X							
SR_NFR_02					X			X							
SR_NFR_03									X						
SR_NFR_04	X	X													
SR_NFR_05							X								
SR_NFR_06										X					
SR_NFR_07											X				
SR_NFR_08											X		X	X	
SR_NFR_09														X	
SR_NFR_10															X
SR_NFR_11					X										
SR_NFR_12						X									

Table 74 - Testing traceability matrix – Non-Functional Requirements

As can be seen in the traceability matrices, each requirement has been tested in both Functional and Non-Functional traceability matrices.

6. Planning and Cost Analysis

This section takes into consideration all the costs that were used to develop the product including material costs and human resources and it shows the scheduling and planning that went into structuring this project.

It also shows the team that worked in the project, their tasks and costs.

Human Resources

To ease the task of measuring the human resources, each phase has been measured in days of development, each of which consist in 8 full hours of work.

Phase	Days of work (8 hours)	Hours
Viability Study	3	24
Basic Requirements	5	40
Analysis	15	120
Design	27	216
Implementation	133	536
Testing	7	56
Documentation	21	168

Table 75 - Days of work per phase

Since the Implementation has a sub-phase structure, a more detailed overview of the implementation sub-phases is described:

Sub Phase	Days of work (8 hours)	Hours
Sensor data acquisition & treatment	67	536
Motor movement	51	408
Structure	10	80
Real time system	5	40

Table 76 - Days of work per sub phase

The human resources needed for the project was only one employee that used a total amount of 1656 hours to develop every step on the process of finishing the project.

To calculate the cost of the hours the following formula was used:

$$Cost = \frac{MonthlySalary}{20\ days * 8\ hours} * DevelpmentHours$$

The following table represents each responsibility and its derived cost based on time of development using the previous formula.

Name and Surname	Position	Hours	Salary	Total Cost
Adrian Pappalardo	Analysis & Design	400	1.500€	3.750€
Adrian Pappalardo	Implementation & Testing	592	1.800€	6.660€
Adrian Pappalardo	Documentation	168	1600€	1.680€

Table 77 – Developers

In total the cost of the human resources totals an amount of 12.090€

A Gantt diagram can be found below describing the time dedicated to each of the different phases of the project.

Gantt Diagram

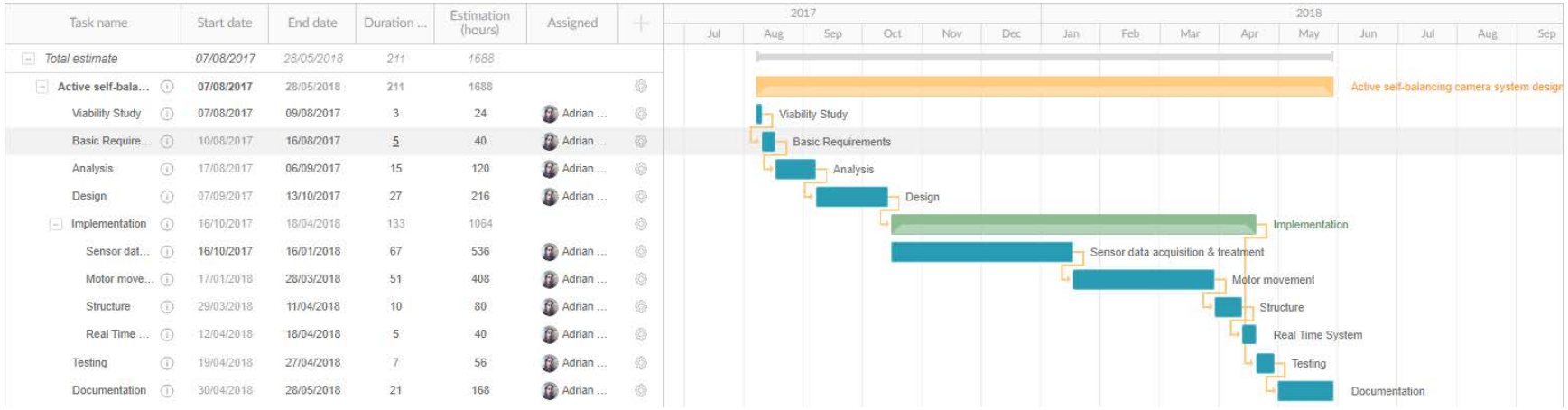


Figure 22 - Gantt diagram

Material Costs

The material costs that are mentioned in this section involve both final components and testing components.

Component	Quantity	Price/Unit	Total Price
Arduino Nano	3	2.7	8.1
MPU6050	3	0.8	2.4
Brushless Motor	4	10	40
Brushless Motor Controller	3	2.7	8.1
Joy Sticks	2	1.3	2.6
Stepper Motor with Controller	5	2	10
3.7v to 5v Converter	2	0.20	0.40
Battery Controller	2	0.60	1.20
Li-Po Battery	5	6	30
External Battery	10	1	10
Aluminum Tube	1	1	1
Aluminum Rod	1	1	1
Aluminum Plate	1	1	1
Roll Bearings	6	1	6
Cables	40	0.1	4

Table 78 - Material costs

The total price for materials during the development was of 103€

In terms of equipment costs, a computer was used during the project, which will be accounted using the following formula:

$$Equipment\ Cost = \frac{ComputerCost}{60\ Months} * Equipment\ usage$$

The same formula applies for the software as well:

Concept	Usage Period	Cost	Total Cost
Sony VAIO Pro Duo 11	12 Months	400	80€
Arduino IDE	12 Months	0	0€
Windows OS	12 Months	40	8€

Table 79 - Equipment Costs

Cost summary

The total cost of the project can be found in the following table:

Concept	Cost
Material Cost	103€
Indirect Costs	3.200€
Equipment Costs	88€
Human Resources	12.090€
IVA (21%)	3.251,01 €
Total	18.732,01 €

Table 80 - Cost summary

7. Conclusions and Future Work

After finalizing the project, it is necessary to use this part of the document to carefully analyze the path that was taken to get to this point and see what objectives were fulfilled.

This section will analyze the project at a personal level and as a project.

Project Conclusions

The project consisted on creating a self-stabilizing camera system using a microcontroller as its core, developing its software, hardware and structure to achieve this goal.

After finishing the project, the main task that the system was set to do was fulfilled, but along the way there were some deviations from the original sketches.

The system fulfills its main task at this point and fulfills all the requirements set at the beginning of the project and the later system and user requirements. But the performance left a bit to be desired. We will be going through all the sections that went great and the ones that did not do so well.

To begin with, the system is reliable, and it fulfilled every expectation that was had at the beginning of this project, it runs with a very good battery life, which was unexpected. It also had the additional feature of being easy to disassemble given the nature of the construction, so moving it to different places to test the system became an easy task.

The addition of a UDB port allows for easy debugging of the system when connected to a computer and it also allowed for easy system updates and tweaks that were added afterwards to enhance some details.

The requirement that forbid external controllers and components was set because there were already boards that worked at a hardware level, but the objective of this project was to develop it at a higher level using software and a microcontroller, and that was successfully completed, except for the motors, there were no stepper motors on the market that would not use their own controllers, and the system did not have enough ports to handle the amount of cables that are needed to run, thus in that case, they do use an external integrated controller to move the motors. This was also decided because the original design using brushless motors came to be extremely difficult and time consuming and it proved that it was a better option to use a controller to complete that task, the only problem was that there are no such controllers for a brushless motor, so the decision to move to a stepper motor was done instead.

The speed of the system is acceptable, but when compared to other gimbals in the market now, the system falls short and it makes the experience very slow, the user’s actions must be slow to compensate for the lack of performance.

It is very incompatible with any other cameras that are not the one used in the requirements. Even if it is stated in the very requirements of the system that there is only a need for that camera to work and that any other camera is indifferent to the matter, comparing it to other systems, they have a bit more strength given the size of the system, but that is only because of the motors themselves and, it is a very feasible upgrade to do.

To summarize the original requirements,

- Make a 3-axis gimbal using only one Arduino Nano.
    - This was successfully achieved fulfilling all the user and system requirements
- Use Brushless motors to operate the system.
    - This proved to be not feasible to achieve for this project, thus this was a requirement that was not met, although it is not included in the user or the system requirements, it was something that I wanted to do.
- Use only one gyroscope/accelerometer sensor without a magnetometer.
    - Even though it was very tempting to use a magnetometer since they offer already curated data, this requirement was proudly fulfilled since all the algorithms to curate the data are done in the Arduino itself.
- Design a structure that supports a Sony QX10 compact camera.
    - This requirement was successfully fulfilled.
- Design a completely energy independent system.
    - The system runs on any kind of power that supports DC 5V through a USB cable, thus any kind of battery enclosure will fulfill the requirement although it also allows for a direct connection to a computer.
- Do not use any pre-built components aside from the Arduino itself, the sensor and the motors.
    - This was not fulfilled; the system uses the motor controllers that came with the stepper motors since it became impossible to develop that section of the code for the steppers since the one for the brushless motors did not work and the time became scarce.

As a conclusion to this section, the system does not work as expected but it does work, and it does fulfill all the user and system requirements, it just does not fulfill the expectations that were set at the beginning.

Personal Conclusions

This project was designed to be a hard challenge to me, I decided to develop something related to robotics and I wanted it to be the most complex system possible, for no other reason than that I wanted to learn as much as possible on an area that I had only basic experience on.

I learned a very broad set of skills based on this project, mostly related to research. The information available for this kind of project was not very broad since there are not a lot of projects like this that are published online, I found that there are a lot of similarities with drones so that is where I focused my research. Other than that, I learned about very low-level Arduino, basic physical interactions with motors and sensors, electronics that I was not very proficient at, and the most important thing was failing. I failed at this project over and over, with every iteration becoming better and better until it finally succeeded, but it was a very hard path to take, but I am very glad that I took it.

Future Work

The system left a lot of room for improvement that can come to a new generation of this device. A few examples will be described below:

- Brushless motors: Even though the project deemed them too hard to implement, there is still the possibility of using them, when comparing the current technologies, they use these motors although on an enclosed environment with hardware controllers, but the possibility is still there, and it will add both power and speed while also being a lot more silent.
- Magnetometer: Since the system does not have a magnetometer, it is a bit hard to reliably calculate the Z axis, thus the addition of this sensor would make the system more reliable in that sense although the difference in price is very high.
- Sensor curation: Even though the system features a good enough curation for the values that are given by the sensor, there is always more room to try new algorithms to enhance this part of the system.

- Add a bidirectional Pan function: This is very easy to do with the current hardware setup of the system, it would be nice to add a slow panning feature in both Z and Y axes of rotation, since the joystick only has two axes.
- Smaller structure: Now the system features a very home-made structure that works fine but it could be improved.

In a general sense, these are some of the features that could be added with the current hardware or adding some new hardware but maintaining the same software structure. There will be more ideas to implement once some of these ideas are fulfilled.

8. References

[1] The Making of a DIY Brushless Gimbal with Arduino, by ArduinoDeXXX, May. 20, 2015
http://www.instructables.com/id/DIY-Brushless-Gimbal-with-Arduino/?ALLSTEPS

[2] Gyro Stabilizer W/ Arduino and Servo, by woojay, Jan. 14, 2016
http://www.instructables.com/id/Gyro-Stabilizer-W-Arduino-and-Servo/

[3] Brushless Gimbal with Arduino, by Jonah Grubb, May. 8 2014
http://www.instructables.com/id/Brushless-Gimbal-with-Arduino/

[4] Spining BLDC(Gimbal) motors at super slooooooow speeds with Arduino and L6234, by Ignas Gramba, Apr. 18, 2015
http://www.berryjam.eu/2015/04/driving-bldc-gimbals-at-super-slow-speeds-with-arduino/

[5] BU-302: Series and Parallel Battery Configurations, Battery University, Nov. 6, 2010
http://batteryuniversity.com/learn/article/serial_and_parallel_battery_configurations

[6] MPU-6000 Register Map, Invensense, Mar 2015
https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf

[7] MPU-6000 Datasheet, Invensense, Mar 2015
https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf

[8] Microcontrollers – Types & Applications, by Tarun Agarwal. 2015
https://www.elprocus.com/microcontrollers-types-and-applications/

[9] Driving a three-phase brushless DC motor with Arduino – Part 1. Theory, elabz, Oct. 30, 2011
http://elabz.com/brushless-dc-motor-with-arduino/

[10] Brushless DC (BLDC) motor with Arduino – Part 2. Circuit and Software, elabz, Dec. 14, 2011
http://elabz.com/bldc-motor-with-arduino-circuit-and-software/

[11] Single axis brushless gimbal with Arduino, by ArduinoDeXXX, May. 20, 2015
https://cdn.instructables.com/ORIG/FDN/FI6T/I9CM26XE/FDNFI6TI9CM26XE.pdf

9. Additional Lectures

IMU gyroscope

http://www.brokking.net/imu.html

https://github.com/jrowberg/i2cdevlib/issues/63

http://forum.arduino.cc/index.php?PHPSESSID=t6lmv7i431eqeaf05ui619q2h6&topic=118937.msg1368958#msg1368958

http://x-io.co.uk/open-source-imu-and-ahrs-algorithms/

https://arduino.stackexchange.com/questions/31552/mpu-6050-angle-drift

https://github.com/TKJElectronics/Example-Sketch-for-IMU-including-Kalman-filter/tree/master/IMU/MPU6050

Brushless Motors

http://www.area52.cc/how-tos/403-pick-the-right-brushless-motor.html

Stepper Motors

http://www.airspayce.com/mikem/arduino/AccelStepper/

http://www.embedded.com/design/mcus-processors-and-socs/4006438/Generate-stepper-motor-speed-profiles-in-real-time

Joystick

https://exploreembedded.com/wiki/Analog_JoyStick_with_Arduino

Videography

Brushless Motors

Brushless Motor Numbers Explained - https://www.youtube.com/watch?v=uLutMoh4Ttg

Controlling Brushless Motors - https://www.youtube.com/watch?v=khvzB0_Wj_U

Brushless without ESC - https://www.youtube.com/watch?v=i8LZBi3EyAE

Arduino Performance - https://www.youtube.com/watch?v=fqEkVcqxtU8

Battery Protection - https://www.youtube.com/watch?v=Qw4psECqpwI

Stepper Motors

Fundamentals on Stepper control - https://channel9.msdn.com/Shows/themakershow/8